

Реферат

Пояснительная записка дипломного проекта содержит 90 страниц, 13 формул, 21 таблицу, 63 иллюстрации, 24 листинга, 16 источников литературы, 6 приложений.

ENTITY FRAMEWORK, ASP.NET CORE, C#, REACT JS, GRAPHQL, HOT CHOCOLATE, SALESFORCE, SALESFORCE SERVICE CLOUD, JETBRAINS RIDER, JETBRAINS WEBSTORM

Целью дипломного проекта является разработка веб-приложения, предназначенного для управления личными финансами.

Пояснительная записка состоит из введения, шести разделов и заключения.

Во введении представлена информация о текущем состоянии в сфере рассматриваемой проблемы, а также поставлены цели и задачи дипломного проектирования.

В первом разделе описаны результаты аналитического обзора литературы.

Во втором разделе представлен обзор средств разработки и обоснование выбора конкретных инструментов.

В третьем разделе приведено описание процесса проектирования и разработки приложения.

В четвертом разделе описано подробное руководство пользователя для разработанного веб-приложения.

В пятом разделе приведена информация, полученная в результате тестирования разработанного веб-приложения.

В шестом разделе представлены результаты расчета себестоимости и отпускной цены разработанного веб-приложения.

В заключении подведены итоги дипломного проектирования и задачи, которые были решены в ходе разработки веб-приложения.

Графическая часть дипломного проекта находится в первых шести приложениях и занимает 1,25 листа А1.

					БГТУ 00.00.ПЗ			
Изм	Лист	№ докум.	Подп.	Дата				
Разраб.	Мацуев И.М.				Реферат	Лит.	Лист	Листов
Пров.	Северинчик Н.А.						1	1
Консульт.	Северинчик Н.А.					74417006, 2021		
Н. контр.	Рыжанкова А.С.							
Утв.	Пацей Н.В.							

Abstract

The explanatory note of the diploma project contains 90 pages of explanatory note, 13 formulas, 21 tables, 63 illustrations, 24 listings, 16 sources used, 6 appendices.

ENTITY FRAMEWORK, ASP.NET CORE, C#, REACT JS, GRAPHQL, HOT CHOCOLATE, SALESFORCE, SALESFORCE SERVICE CLOUD, JETBRAINS RIDER, JETBRAINS WEBSTORM

The purpose of the graduation project is to create a web-application for maintaining a user personal finance.

The explanatory note consists of an introduction, seven sections, conclusion, and a list of references.

The introduction sets out the goal and objectives of the project.

The first section describes the results of the analytical literature review.

The second section provides an overview of the development tools and the rationale for choosing specific tools.

The third section describes the design and development process for an application.

The fourth section describes a detailed user guide for the developed web application.

The fifth section contains information obtained as a result of testing the developed web application.

The sixth section presents the results of calculating the cost and selling price of the developed web application.

In the conclusion, the results of the diploma design and the tasks that were solved during the development of the web application are summed up.

The graphic part of the diploma project is in six appendices and takes 1.25 sheets A1.

					БГТУ 00.00.ПЗ		
Изм.	Лист	№ докум.	Подп.	Дата			
Разраб.	Мацуев И.М.				Abstract	Лист	Листов
Пров.	Северинчик Н.А.					1	1
Консульт.	Северинчик Н.А.					74417006, 2021	
Н. контр.	Рыжанкова А.С.						
Утв.	Пацей Н.В.						

Содержание

Введение	8
1 Аналитический обзор аналогов и технических средств	9
1.1 Система учета личных финансов	9
1.2 Обзор аналогов разрабатываемого веб-приложения	9
1.2.1 <i>Money Lover</i>	10
1.2.2 <i>Spendee</i>	11
1.2.3 <i>Wallet</i>	11
1.3 Обоснование технических средств программирования.....	13
1.3.1 Платформа <i>ASP.NET Core</i>	13
1.3.2 Язык программирования <i>C#</i>	13
1.3.3 Фреймворк <i>Entity Framework</i>	14
1.3.4 <i>GraphQL</i> и библиотека <i>Hot Chocolate</i>	15
1.3.5 <i>ReactJS</i> и <i>Apollo Client</i>	15
1.3.6 <i>JetBrains WebStorm</i>	16
1.3.7 <i>JetBrains Rider</i>	17
1.3.8 Платформа <i>Salesforce</i>	17
1.4 Вывод по разделу	18
2 Проектирование веб-приложения	19
2.1 Основные технические требования к разработке.....	19
2.2 Проектирование базы данных	19
2.2.1 Модель <i>User</i>	20
2.2.2 Модель <i>Setting</i>	20
2.2.3 Модель <i>Language</i>	21
2.2.4 Модель <i>Currency</i>	21
2.2.5 Модель <i>TransactionType</i>	21
2.2.6 Модель <i>BugReport</i>	22
2.2.7 Модель <i>UserEmailVerificationCode</i>	22
2.2.8 Модель <i>UserRefreshToken</i>	23
2.2.9 Модель <i>UserSetting</i>	23
2.2.10 Модель <i>UserNotification</i>	23
2.2.11 Модель <i>Category</i>	24
2.2.12 Модель <i>CurrencyExchangeRate</i>	24
2.2.13 Модель <i>Wallet</i>	24
2.2.14 Модель <i>Transaction</i>	25
2.2.15 Модель <i>RegularTransaction</i>	25
2.2.16 Контекст <i>CashSchedulerContext</i>	26

					БГТУ 00.00.ПЗ						
Изм	Лист	№ докум.		Подп.	Дата						
Разраб.	Мацуев И.М.				Содержание	Лит.		Лист		Листов	
Пров.	Северинчик Н.А.							1		3	
Консульт.	Северинчик Н.А.					74417006, 2021					
Н. контр.	Рыжанкова А.С.										
Утв.	Пацей Н.В.										

2.3 Структура приложения.....	27
2.4 Вывод по разделу	29
3 Разработка веб-приложения.....	30
3.1 Описание разработанных <i>GraphQL</i> резолверов	30
3.1.1 Резолверы модели <i>User</i>	30
3.1.2 Резолверы модели <i>Category</i>	33
3.1.3 Резолверы модели <i>Wallet</i>	35
3.1.4 Резолверы модели <i>Transaction</i>	37
3.1.5 Резолверы модели <i>RegularTransaction</i>	39
3.1.6 Резолверы модели <i>UserSetting</i>	40
3.1.7 Резолверы модели <i>UserNotification</i>	42
3.2 Настройка и установка библиотеки <i>Hot Chocolate</i>	44
3.3 Реализация интеграции с <i>Salesforce</i>	45
3.3.1 Реализация схемы издатель-подписчик	46
3.3.2 Реализация сервиса для общения с <i>Salesforce REST API</i>	47
3.3.3 Настройка <i>Salesforce</i> организации.....	49
3.4 Вывод по разделу	54
4 Руководство пользователя	55
4.1 Страница входа	55
4.2 Форма отправки обратной связи	58
4.3 Главная страница	58
4.4 Страница просмотра и редактирования кошельков	64
4.5 Страница просмотра и изменения категорий	68
4.6 Страница с детальным описанием транзакций.....	70
4.7 Страница просмотра и изменения настроек пользователя.....	74
4.8 Выводы по разделу	76
5 Тестирование веб-приложения	77
5.1 Unit тестирование	77
5.2 Ручное тестирование	79
5.2.1 Регистрация и вход в аккаунт.....	79
5.2.2 Создание транзакции.....	80
5.2.3 Создание перевода.....	81
5.3 Вывод по разделу	82
6 Экономическое обоснование цены веб-приложения	83
6.1 Общая характеристика разрабатываемого веб-приложения	83
6.2 Исходные данные и маркетинговый анализ	83
6.3 Методика обоснования цены.....	84
6.3.1 Объем веб-приложения.....	85
6.3.2 Основная заработная плата.....	86
6.3.3 Дополнительная заработная плата.....	86
6.3.4 Отчисления в Фонд социальной защиты населения.....	87
6.3.5 Расходы на материалы	87
6.3.6 Расходы на оплату машинного времени	87
6.3.7 Прочие прямые затраты	88
6.3.8 Накладные расходы.....	88

6.3.9 Сумма расходов на разработку веб-приложения	88
6.3.10 Расходы на сопровождение и адаптацию	88
6.3.11 Полная себестоимость.....	89
6.4 Вывод по разделу	89
Заключение	90
Список использованных источников.....	91
ПРИЛОЖЕНИЕ А Диаграмма использования	92
ПРИЛОЖЕНИЕ Б Логическая схема базы данных.....	93
ПРИЛОЖЕНИЕ В Диаграмма классов.....	94
ПРИЛОЖЕНИЕ Г Скриншоты работы программы	95
ПРИЛОЖЕНИЕ Д Блок-схема алгоритма создания транзакции.....	96
ПРИЛОЖЕНИЕ Е Блок-схема алгоритма создания перевода	97

Введение

Чтобы стать финансово независимым человеком, нужно уделять внимание не только тому, сколько вы зарабатываете, но и тому, сколько тратите. Человек постоянно старается извлечь максимальную выгоду из любой сделки и любого случая, поэтому всегда пользовались спросом приложения и утилиты, позволяющие улучшить уже существующие процессы менеджмента. Один из таких процессов – это подход к мониторингу состояния человека или компании. Реализовать такой подход в настоящее время достаточно просто — необходимо начать отслеживать свои расходы и доходы с помощью одного из специальных приложений для мобильных телефонов или персональных компьютеров.

В сети существует огромное количество подобного рода приложений, однако очень многие из них не подходят для повседневного использования или попросту неудобны. Этот проект призван перенять лучшие качества и нивелировать худшие качества аналогов с целью создания приложения, которое было бы лучше других как со стороны пользователя, так и со стороны бизнеса.

Целью дипломного проекта является проектирование и разработка веб-приложения для управления личными финансами пользователей, включающего серверную и клиентскую части.

Для достижения поставленной цели необходимо:

- произвести анализ аналогичных продуктов;
- составить набор требований к разрабатываемому веб-приложению;
- спроектировать архитектуру приложения и базы данных;
- разработать серверную и клиентскую часть приложения;
- произвести тестирование работоспособности приложения и определить соответствие поставленным требованиям.

Выполнение всех перечисленных выше задач в результате реализации проекта должно свидетельствовать об успешности дипломного проектирования.

					БГТУ 00.00.ПЗ								
Изм	Лист	№ докум.		Подп.	Дата								
Разраб.		Мацуев И.М.				Введение				Лит.	Лист	Листов	
Пров.		Северинчик Н.А.										1	1
Консульт.		Северинчик Н.А.								74417006, 2021			
Н. контр.		Рыжанкова А.С.											
Утв.		Пацей Н.В.											

1 Аналитический обзор аналогов и технических средств

1.1 Система учета личных финансов

Система учета личных финансов – это вид программного обеспечения, которое предоставляет пользователю возможность фиксировать и планировать все необходимые финансовые операции, в том числе траты и доходы. Часто предоставляет визуальное представление бюджета за определенный период времени.

Раньше зачастую самым главным инструментом для ведения личной бухгалтерии служил *Excel*, так как это очень мощный инструмент и действительно подходит для расчета различного рода финансовых операций и подвода итогов. Однако в дальнейшем курс ушел в сторону более понятного интерфейса взаимодействия для конечного пользователя, именно поэтому появились приложения, изначалью заточенные под задачу контроля финансов.

Подобного рода системы позволяют контролировать источники доходов, не упускать из виду возможные личные финансовые риски и собирать информацию за промежуток времени, без надобности держать это все в голове. Также, они могут информировать вас об надвигающихся тратах, давать советы по финансовому менеджменту. Существуют как приложения для личного пользования, так и корпоративные продукты, как например – для учета трат сотрудника в командировке.

На данный момент существует большое количество реализаций подобных систем, такие как *MoneyLover*, *Spendee*, *Mint.com*, *GnuCash* и т.д. Из-за большой конкуренции на рынке, каждое из этих приложений старается дать что-то уникальное пользователю, начиная от привлекательного интерфейса, заканчивая максимально подробной статистикой или возможностью привязки банковских карт и счетов. Каждое из этих приложений разработано отдельной группой разработчиков, имеют отдельные серверы и протоколы, отличаются своими правилами, особенностями и возможностями. Таким образом необходимо перенять все лучшее и избежать неудобных моментов и ошибок из существующих на сегодняшний день приложений учета личных финансов пользователей.

1.2 Обзор аналогов разрабатываемого веб-приложения

В мире существует множество приложений учета финансов, с различного рода функциями, как платных, так и бесплатных. Но кроме десктопных или мобильных приложений популярны также веб-версии, рассмотрим самые распространенные из них.

					БГТУ 01.00.ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата				
Разраб.	Мацуев И.М.				1 Аналитический обзор аналогов и технических средств	Лит.	Лист	Листов
Пров.	Северинчик Н.А.						1	10
Консульт.	Северинчик Н.А.					74417006, 2021		
Н. контр.	Рыжанкова А.С.							
Утв.	Пацей Н.В.							

1.2.1 Money Lover

В качестве одного из прототипов было выбрано приложение *MoneyLover*. В отличие от многих других, *MoneyLover* — не просто про учет расходов или планирование бюджета. Программа охватывает практически все возможные сценарии использования денег, а стало быть — большинство ситуаций, которые могут возникнуть на жизненном пути.

Существуют версии приложения для всех платформ, включая веб-версию. Пользовательский интерфейс приложения представлен на рисунке 1.1.

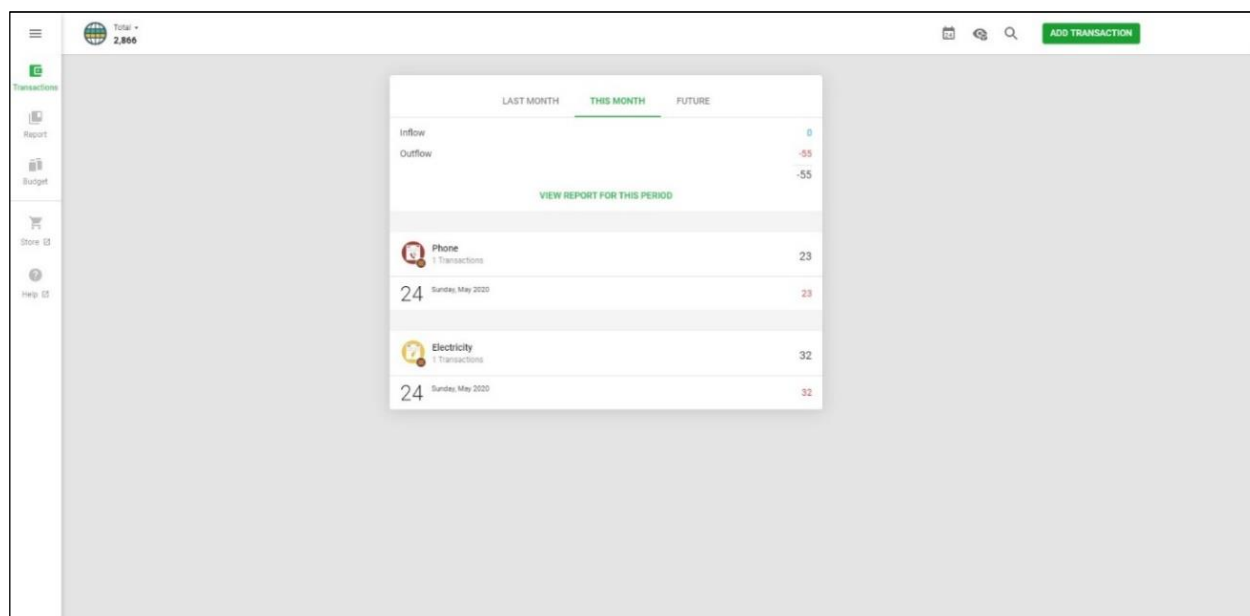


Рисунок 1.1 – Пользовательский интерфейс приложения *MoneyLover*

Приложение предназначено для учета и анализа собственных расходов и доходов. Приложение имеет довольно красивый и понятный любому пользователю интерфейс и функционал.

Достоинства приложения:

- богатые функциональные возможности;
- продвинутое управление счетами и категориями;
- функция сканирования чеков;
- учет долгов и постановка финансовых целей;
- учет задолженности и доступного лимита по кредитным картам;
- совместный учет финансов;
- синхронизация транзакций с *PayPal*;
- автоматическая и ручная конвертация валюты;
- информативный виджет.

Недостатки приложения:

- проблемы с синхронизацией информации между девайсами;
- рекламные окна, мешающие работе;
- проблемы с переводом в русскоязычной версии.

1.2.2 Spendee

Spendee выделяется на фоне остальных приложений красивым интерфейсом. Здесь нет скучных таблиц, напоминающих о скучной бухгалтерской работе. Вместо этого *Spendee* предлагает удобный и привлекательный *UI*, чем-то напоминающий ленту в соцсети. Ваши доходы и расходы будут представлены в виде красивой инфографики, так что вы сможете с легкостью понять, что происходит с вашими деньгами.

Приложение *Spendee* существует с версиями под все мобильные устройства, в том числе и веб-версия.

Отличительной особенностью является возможность привязки банковских счетов для автоматического заполнения транзакций по счетам.

Пользовательский интерфейс приложения продемонстрирован ниже на рисунке 1.2.

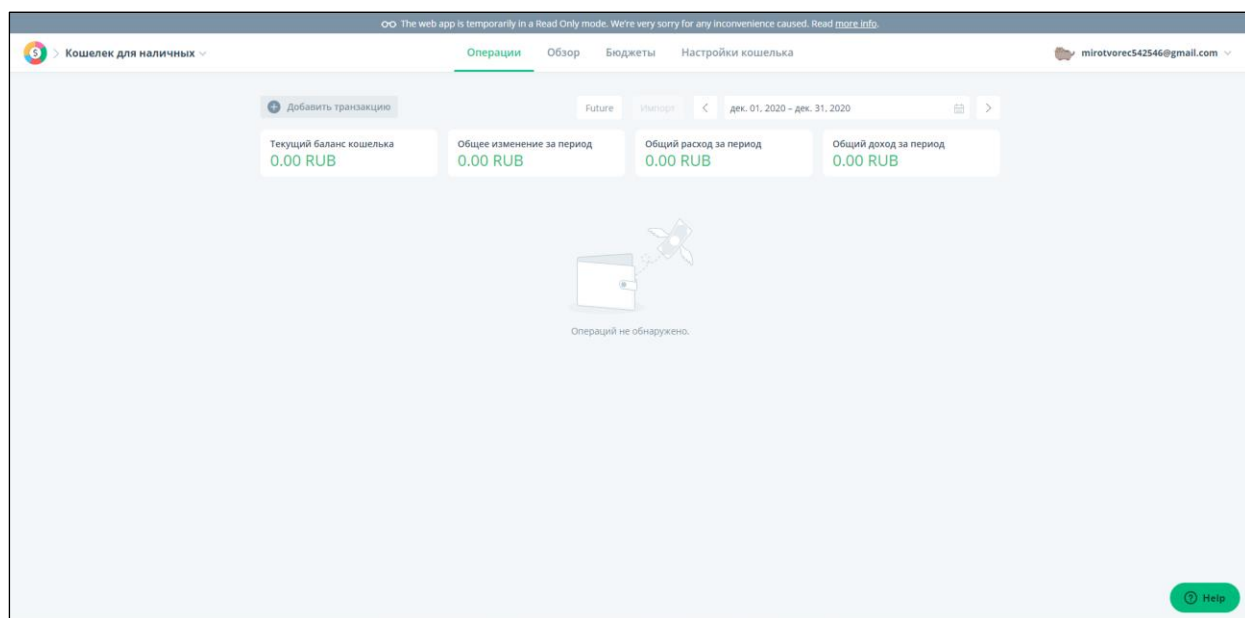


Рисунок 1.2 – Пользовательский интерфейс приложения *Spendee*

Так же приложение имеет возможность создавать кошельки в разной валюте, добавлять транзакции вручную, а так же просматривать статистику за определенный промежуток времени.

Достоинства приложения:

- в приложении можно спланировать расходы по каждой отдельной категории;
- можно вести совместный бюджет с другими аккаунтами.

Недостатки приложения:

- нет возможности привязать к аккаунту больше одного кошелька.

1.2.3 Wallet

Это очень популярное приложение для контроля расходов. Оно поддерживает различные валюты и помогает точно планировать бюджет. Использовать его можно бесплатно, но если вы оформите платную подписку, то

сможете синхронизировать свои банковские транзакции между несколькими устройствами и автоматически классифицировать их, а также добавлять несколько банковских счетов (в бесплатной версии можно использовать не больше трех).

Пользовательский интерфейс приложения показан на рисунке 1.3.

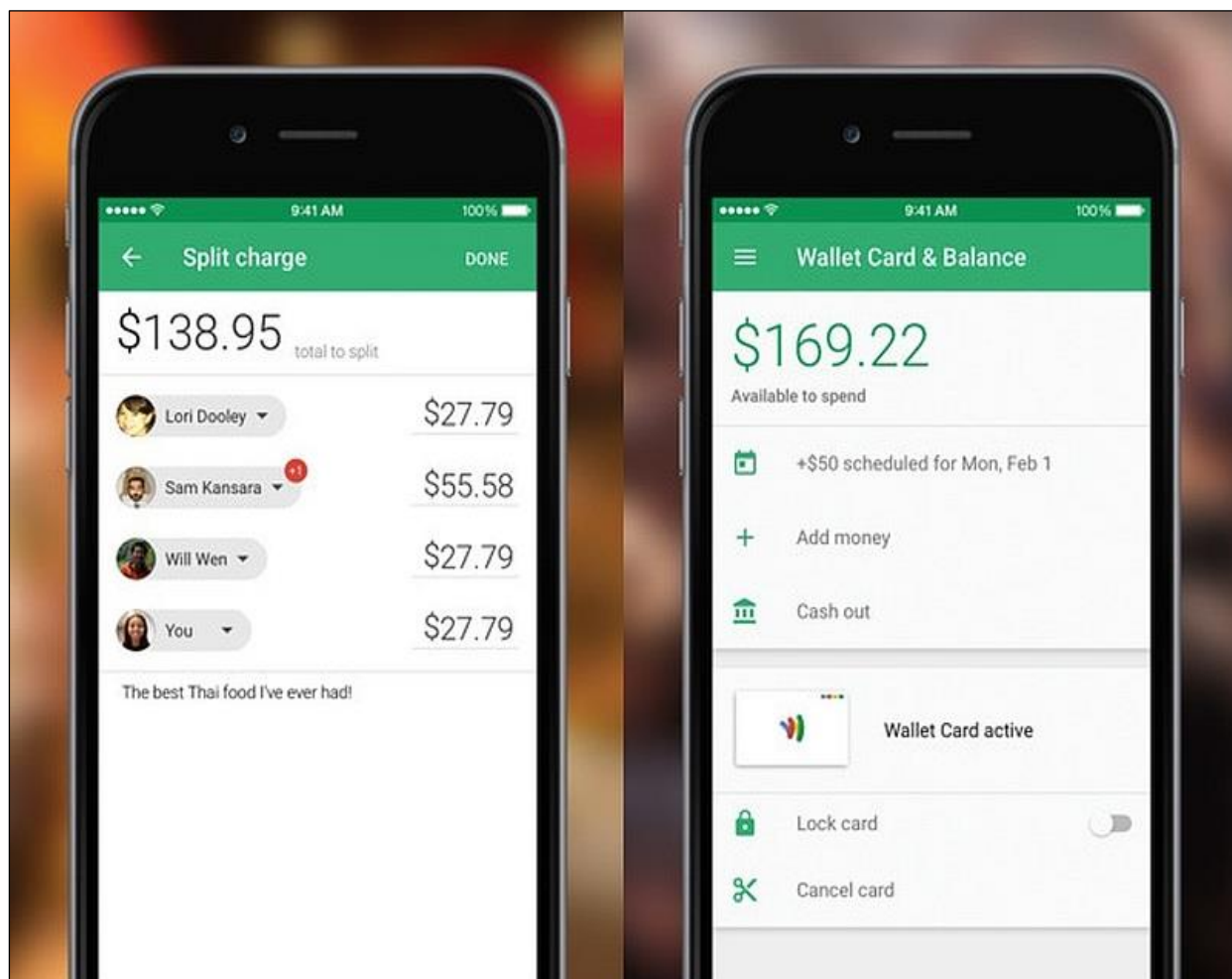


Рисунок 1.3 – Пользовательский интерфейс приложения *Wallet*

Помимо этого, приложение может создавать шаблоны платежей и списки покупок, а также экспортировать данные о ваших доходах и расходах в различные форматы.

Достоинства приложения:

- приложение синхронизируется с банковскими операциями;
- планировать бюджет для определенных покупок помогает функция «создать цель»;
- приложение показывает актуальные курсы валют.

Недостатки приложения:

- нет возможности создавать цели в разных валютах.

1.3 Обоснование технических средств программирования

1.3.1 Платформа ASP.NET Core

Для написания сервера приложения была выбрана платформа *ASP.NET Core*. Для обоснования выбора отметим отличительные особенности данной технологии. *ASP.NET Core* – фреймворк с открытым исходным кодом для платформы *.NET*. Данный фреймворк предоставляет огромный набор инструментов для проектирования различного рода серверов, сервисов и других веб приложения на базе платформы *.NET*. На данный момент более широко используется версия для *.NET Core* по ряду причин: платформа с открытым исходным кодом, кроссплатформенность, возможность *self*-хостинга и многое другое.

.NET Core основана на *.NET Framework*. Платформа *.NET Core* отличается от нее модульностью, кроссплатформенностью, возможностью применения облачных технологий, и тем, что в ней произошло разделение между библиотекой *CoreFX* и средой выполнения *CoreCLR*. Несмотря на то, что это новый фреймворк, построенный на новом веб-стеке, он обладает высокой степенью совместимости концепций с *ASP.NET*. Приложения *ASP.NET Core* поддерживают параллельное управление версиями, при котором разные приложения, работающие на одном компьютере, могут ориентироваться на разные версии *ASP.NET Core*. Это было невозможно в предыдущих версиях *ASP.NET*.

Разрабатывать приложения на основе *ASP.NET Core* можно с помощью таких языков программирования как *VB* и *C#*. В моем случае я вел разработку на языке *C#*, т.к. он имеет привычный *C*-подобный синтаксис, множество функций для удобной разработки на уровне языка, а так же огромную фан базу, что позволило быстро находить ответы на проблемы в процессе реализации проекта.

Для построения серверной части приложения я выделил для себя *GraphQL* подход. Принцип разработки *GraphQL API* отличается от привычного *REST*, однако предоставляет массу преимуществ в разработке и поддержания *API*.

С помощью *NuGet* пакетного менеджера была найдена библиотека *HotChocolate* для построения *GraphQL* сервиса на базе *ASP.NET Core*. Библиотека достаточно хорошо документированна и активно поддерживается самой компанией *Microsoft*.

1.3.2 Язык программирования C#

C# – современный объектно-ориентированный и типобезопасный язык программирования. *C#* позволяет разработчикам создавать множество типов безопасных и надежных приложений, работающих в экосистеме *.NET*.

C# относится к широко известному семейству языков *C*, и покажется хорошо знакомым любому, кто работал с *C*, *C++*, *Java* или *JavaScript*. Здесь представлен обзор основных компонентов языка *C# 8* и более ранних версий.

C# – это объектно- и компонентно-ориентированный язык программирования, который предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. С момента создания язык *C#* обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО.

Вот лишь несколько функций языка *C#*, которые позволяют создавать надежные и устойчивые приложения:

- Сборка мусора – автоматически освобождает память, занятую недоступными неиспользуемыми объектами.
- Типы, допускающие значение *null* – обеспечивают защиту от переменных, которые не ссылаются на выделенные объекты.
- Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и восстановлению после них.
- Лямбда-выражения поддерживают приемы функционального программирования.
- Синтаксис *LINQ* создает общий шаблон для работы с данными из любого источника.
- Все типы *C#*, включая типы-примитивы, такие как *int* и *double*, наследуют от одного корневого типа *object*. Все типы используют общий набор операций, а значения любого типа можно хранить, передавать и обрабатывать схожим образом. Более того, *C#* поддерживает как определяемые пользователями ссылочные типы, так и типы значений.
- Динамическое выделение объектов и хранение упрощенных структур в стеке. *C#* поддерживает универсальные методы и типы, обеспечивающие повышенную безопасность типов и производительность.
- Итераторы, которые позволяют разработчикам классов коллекций определять пользовательские варианты поведения для клиентского кода.

1.3.3 Фреймворк Entity Framework

Entity Framework представляет собой специальную объектно-ориентированную технологию для работы с данными. Обычно мы оперируем таблицами, индексами, первичными и внешними ключами. Но при использовании *Entity Framework*, мы уже работаем с объектами.

Центральной концепцией *Entity Framework* является понятие сущности или *entity*. Сущность представляет набор данных, ассоциированных с определенным объектом. При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных. *Entity Framework* предполагает три способа взаимодействия с базой данных:

- *database first*: *Entity Framework* создает набор классов, соответствующий модели конкретной базы данных;
- *model first*: разработчик создает модель базы данных, по которой *Entity Framework* создает реальную базу данных на сервере;
- *code first*: разработчик создает класс модели данных, которые будут храниться в базе данных, а затем *Entity Framework* по этой модели генерирует базу данных и ее таблицы.

1.3.4 GraphQL и библиотека Hot Chocolate

GraphQL – это язык запросов и манипулирования данными с открытым исходным кодом для *API*, а также среда выполнения для выполнения запросов с существующими данными.

С помощью запросов *GraphQL* получает необходимые данные с сервера. *Query* в *GraphQL* – аналог *GET* в *REST*. Запросы – строки, которые отправляются в теле *HTTP POST*. *Query* описывает данные, которые необходимо получить с сервера. В ответ на этот запрос сервер присылает данные в формате *JSON*. Структура ответа соответствует структуре запроса. Успешные операции возвращают *JSON* с ключом и данными, а неуспешные возвращают *JSON* с ключом и сообщением об ошибке. Благодаря этому удобно обрабатывать ошибки на стороне клиента. *Query* – корневой тип (*root type*), так как он сравнивается с операциями.

Mutation – еще один корневой тип. С его помощью можно добавлять данные в БД. *Mutation* – аналог *POST* и *PUT* в *REST*.

Subscription — третий тип операций в *GraphQL*. С его помощью клиент слушает изменения в БД в режиме реального времени. Под капотом подписки используют вебсокеты. Подобный запрос можно использовать для обновления количества информации в режиме реального времени в соответствующем интерфейсе, например, в форме с результатами голосования на сайте.

Реализации самих методов запросов называются резолверами.

Для работы с *GraphQL* в *ASP.NET Core* существует *NuGet* пакет под названием *Hot Chocolate*. Эта одна из самых распространенных библиотек для работы с *GraphQL* в *C#*. Она постоянно улучшается и имеет хорошую документацию.

Одно из достоинств этой библиотеки в том, что она очень хорошо интегрирована с *ASP.NET*, и *DI* контейнером *.NET Core*. Авторизация и аутентификация настраивается теми же стандартными инструментами, а также имеется интеграция с *Entity Framework*, что позволяет экономить много времени на написании кода.

1.3.5 ReactJS и Apollo Client

Библиотека *ReactJS* – это одна из самых популярных библиотек для создания сложных клиентских приложений. Представляет собой *JavaScript* библиотеку для построения пользовательских интерфейсов. Данная библиотека работает с виртуальным *DOM*, самостоятельно отслеживает изменения в состоянии приложения и выполняет перерисовку лишь той тех элементов страницы, которые были изменены, что является несомненным плюсом для производительности.

ReactJS основывается на так называемых компонентах – это классы или функции, позволяющие описывать элементы *DOM*, производить различные операции над ними в ходе жизненного цикла компонента.

Компоненты *ReactJS* являются переиспользуемыми, поскольку представляют собой обычные классы, унаследованные от *React.Component*. При грамотном проектировании это позволяет исключить дублирование кода, обеспечить гибкость и далее легкую поддержку готового приложения. Для *ReactJS* существует

множество оберток, библиотек и готовых решений, которые позволяют просто и быстро начать реализацию нового проекта. Библиотека является самой популярной из рассматриваемых.

Причины использовать *React*:

- отлично подходит для командной разработки, строгое соблюдение *UI*, и шаблона рабочего процесса;
- код читабельный и прост в сопровождении;
- разработка *UI* на основе отдельных компонентов – это будущее веб-разработки и чем раньше это поймет разработчик – тем лучше для него.

Одна из главных особенностей *ReactJS* – это свобода действий, существует огромное количество подходов к построению приложений с его помощью (*ApolloClient*, *Redux*, *mobx* и другие).

К минусам *ReactJS* можно отнести сложность в освоении, поскольку для создания действительно хороших приложения с его использованием необходимо хорошее понимание его концепций.

Платформа *Apollo* это реализация *GraphQL*, помогающая переправлять данные из облака к *UI*. Эта платформа может быть наложена в виде дополнительного слоя на уже существующие сервисы, включая *REST API* и базы данных.

Apollo Client – это клиентская библиотека для *Apollo GraphQL*. Это полнофункциональный кэширующий клиент *GraphQL* с интеграцией с *React*.

Apollo Client позволяет пользователям легко создавать *UI*-компоненты, которые получают данные через *GraphQL*. Благодаря декларативному подходу к извлечению данных вся логика для извлечения пользовательских данных, отслеживания загрузки и состояний ошибок и обновления *UI* инкапсулирована в одном компоненте *Query*, который может быть составлен из презентационных компонентов. Весь этот подход во многих случаях может упростить процесс разработки.

1.3.6 JetBrains WebStorm

JetBrains WebStorm — интегрированная среда разработки на *JavaScript*, *CSS* & *HTML* от компании *JetBrains*, разработанная на основе платформы *IntelliJ IDEA*.

WebStorm обеспечивает автодополнение, анализ кода на лету, навигацию по коду, рефакторинг, отладку, и интеграцию с системами управления версиями. Важным преимуществом интегрированной среды разработки *WebStorm* является работа с проектами (в том числе, рефакторинг кода *JavaScript*, находящегося в разных файлах и папках проекта, а также вложенного в *HTML*). Поддерживается множественная вложенность (когда в документ на *HTML* вложен скрипт на *Javascript*, в который вложен другой код *HTML*, внутри которого вложен *Javascript*) — то есть в таких конструкциях поддерживается корректный рефакторинг. Данная среда разработки доступна для *Windows*, *OS X* и *Linux*.

Особенности данного продукта:

1. Сборка приложений на *Node.js*.
2. Модификация *js* файлов с одновременным просмотром результатов.
3. Рефакторинг кода.

4. Статический анализатор кода (*Lint*), позволяющий находить проблемы производительности, несовместимости версий и другое.
5. Интеграция с системами управления версиями.
6. Возможность расширения плагинами.
7. Отладка кода на *JavaScript*.

1.3.7 JetBrains Rider

JetBrains Rider – кроссплатформенная интегрированная среда разработки программного обеспечения для платформы *.NET*, разрабатываемая компанией JetBrains. Поддерживаются языки программирования *C#*, *VB.NET* и *F#*. Проект анонсирован в январе 2015 года. В его основе лежит другой продукт *JetBrains* – *ReSharper*. Среда поддерживает платформы *.NET Framework*, *.NET Core* и *Mono*. Работает на операционных системах *Windows*, *MacOS*, *Linux*. Дизайн среды ориентирован на продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как *Rider* берет на себя выполнение рутинных операций.

Среди прочих возможностей, среда хорошо совместима со многими популярными свободными инструментами разработчиков, такими как *VCS*, *Entity Framework*, *NuGet* и *XUnit*.

Rider поддерживает *.NET Framework*, новую платформу *.NET Core* и проекты на основе *Mono*. *IDE* позволяет разрабатывать десктопные приложения, *.NET*-сервисы и библиотеки, игры на движке *Unity*, мобильные приложения *Xamarin*, веб-приложения *ASP.NET* и *ASP.NET Core*.

Rider предоставляет более 2200 инспекций кода, сотни контекстных действий и рефакторингов, заимствованных из *ReSharper*, в сочетании с продвинутой функциональностью сред разработки на основе платформы *IntelliJ*. Несмотря на большой набор функций, *Rider* – быстрая и отзывчивая *IDE*, поэтому мой выбор пал именно на эту *IDE*.

1.3.8 Платформа Salesforce

Salesforce – это *CRM*-система, которая предназначена для управления самыми разными видами бизнес-процессов, отношениями с клиентами, аналитикой, маркетингом, продажами и прочее. Говоря простым языком, это платформа, помогающая бизнесу и компаниям увеличить прибыль и уменьшить траты за счет отлаживания процессов общения с клиентами и персоналом.

Работает *Salesforce CRM* по модели *SaaS* (форме вычислений в облаке). Пользователям *CRM*-системы предоставляется готовое программное обеспечение, доступ к которому открыт через браузер или мобильное приложение. *CRM*-система ориентирована на малый, средний и крупный бизнес. Предлагает все необходимое для поиска, удержания клиентской базы, совершения сделки и обработки входящих запросов клиентов.

Salesforce содержит в себе множество инструментов для автоматизации бизнес процессов, а также большое количество сервисов специфичных для определенных сфер бизнеса. Для дипломного проекта из множества облаков *Salesforce* я использовал *Service Cloud*.

Service Cloud – функциональная часть *Salesforce* для обеспечения процессов обслуживания и поддержки клиентов. Система позволяет компаниям-подписчикам построить сайт самообслуживания и поддержки своих клиентов, специалисты по поддержке получают возможность взаимодействовать с клиентами, вести базы знаний, управленцы компании-подписчика получают инструменты для анализа эффективности сервисных процессов. Таким образом это облако поможет в реализации уровня центра поддержки пользователей, которые оставляют отзывы или хотят сделать какое-либо предложение для улучшения приложения.

1.4 Вывод по разделу

В данном разделе было проведено исследование существования аналогов и прототипа. По результатам исследования у создаваемого веб-приложения было обнаружено несколько аналогов, но каждый из них, несмотря на достоинства и особенности, имеет определенные существенные недостатки. Избегание недостатков и внедрение дополнительных функций в приложение должно помочь стать ему более актуальным на рынке среди подобного рода аналогов.

Также, исходя из анализа, были подобраны наиболее актуальные и подходящие для задачи инструменты: платформы и среды разработки, приведено описание данных инструментов и аргументирован выбор в пользу каждого из них.

В данном дипломном проекте разработка ведется над веб-приложением, исходя из того вывода, что в таком случае пользоваться приложением смогут все, пускай и с меньшим удобством, нежели при реализации на какую то конкретную платформу, будь то десктоп или мобильное устройство.

Исходя из анализа решений на рынке была спроектирована диаграмма вариантов использования. Диаграмма вариантов использования данного веб-приложения представлена в приложении А.

Суммируя все вышеперечисленное можно сказать, что работа, проделанная по аналитическому обзору литературы, является крепким фундаментом для разрабатываемого веб-приложения. На его основе можно приступить к проектированию и разработке моделей базы данных, резолверов и представлений веб-приложения.

2 Проектирование веб-приложения

2.1 Основные технические требования к разработке

Целью дипломного проекта является проектирование и разработка веб-приложения для управления личными финансами пользователей, включающего серверную и клиентскую части.

Основными требованиями являются:

- разработать архитектуру базы данных;
- обеспечить регистрацию пользователя;
- обеспечить возможность восстановления доступа к аккаунту пользователя;
- обеспечить авторизацию пользователя;
- обеспечить возможность отправки отзывов о приложении;
- обеспечить регистрацию одиночных и повторяющихся денежных транзакций пользователя;
- обеспечить создание и управление кошельками и категориями транзакций;
- обеспечить визуализацию статистики транзакций пользователя за заданный период времени;
- реализовать механизм оповещения пользователя в приложении;
- обеспечить возможность настройки работы приложения пользователя;
- реализовать сбор и визуализацию статистики использования приложения пользователями.

Таким образом, цель данного проекта заключается в разработке приложения, удовлетворяющего вышесказанным критериям.

2.2 Проектирование базы данных

При проектировании классического приложения, работа начиналась бы с проектирования базы данных. В данном дипломном проекте был использован подход *Code First*. Это значит, что изначально разрабатываются классы приложения (модели данных) и затем, на их основе фреймворк *Entity Framework* создаст все необходимые таблицы в базе данных. Такой подход значительно сокращает время проектирования и реализации базы данных, а также не требует от разработчика глубоких познаний по тематике проектирования и разработки баз данных. Также, при проектировании моделей базы данных, каждой из них были присвоены базовые валидации полей, которые предоставляет *ASP.NET*. С их помощью, можно избежать дополнительного написания кода с целью валидации сущностей – это можно сделать буквально одной строкой специальными атрибутами.

					БГТУ 02.00.ПЗ		
Изм	Лист	№ докум.	Подп.	Дата			
Разраб.	Мацуев И.М.				2 Проектирование веб-приложения	Лит.	Лист
Пров.	Северинчик Н.А.						1
Консульт.	Северинчик Н.А.						11
Н. контр.	Рыжанкова А.С.					74417006, 2021	
Утв.	Пацей Н.В.						

Еще одно достоинство *Entity Framework* это возможность легко манипулировать миграциями. Это очень важно, т.к. в процессе разработки часто меняется необходимость в тех или иных полях, таблицах и связях. Миграции помогают сохранить историю изменений базы данных и избежать проблем при запуске приложения на другом компьютере.

2.2.1 Модель User

Данная модель описывает пользователя. Все поля модели описаны в таблице 2.1. Сама модель создается автоматически при первой регистрации пользователя в приложении.

Таблица 2.1 – Модель *User*

Поле	Назначение
<i>Id</i>	Поле, в котором будет храниться <i>Id</i> пользователя (первичный ключ);
<i>FirstName</i>	Поле, в котором хранится имя пользователя (не обязательное);
<i>LastName</i>	Поле, в котором хранится фамилия пользователя (не обязательное);
<i>Email</i>	Поле, в котором хранится электронный почтовый адрес пользователя;
<i>Password</i>	Поле, в котором хранится хэш пароля пользователя;

При реализации данной модели было создано пять полей в таблице «*User*», в которой в качестве первичного ключа выступает «*Id*».

2.2.2 Модель Setting

Данная модель содержит в себе данные существующих настроек в системе. Все поля таблицы описаны в таблице 2.2.

Таблица 2.2 – Модель *Setting*

Поле	Назначение
<i>Name</i>	Поле, в котором будет храниться имя настройки (первичный ключ);
<i>Label</i>	Поле, в котором будет храниться заголовок настройки;
<i>UnitName</i>	Поле, в котором будет храниться имя раздела настроек;
<i>SectionName</i>	Поле, в котором будет храниться имя секции раздела настроек;

Окончание таблицы 2.2

Поле	Назначение
<i>ValueType</i>	Поле, в котором будет храниться тип значения настройки (<i>Text</i> , <i>Checkbox</i> , <i>Custom</i>).
<i>Description</i>	Поле, в котором будет храниться описание настройки.

При реализации данной модели было создано шесть полей в таблице «*Setting*», в которой в качестве первичного ключа выступает «*Name*».

2.2.3 Модель *Language*

Данная модель содержит в себе языки, которые будут доступны пользователю для выбора. Поля таблицы описаны в таблице 2.3.

Таблица 2.3 – Модель *Language*

Поле	Назначение
<i>Abbreviation</i>	Сокращенное имя языка (первичный ключ);
<i>Name</i>	Полное имя языка;
<i>IconUrl</i>	Ссылка на иконку для данного языка.

При реализации данной модели было создано три поля в таблице «*Language*», в которой в качестве первичного ключа выступает «*Abbreviation*».

2.2.4 Модель *Currency*

Данная модель содержит в себе валюты, которые будут доступны пользователю для выбора при создании кошельков. Поля таблицы описаны в таблице 2.4.

Таблица 2.4 – Модель *Currency*

Поле	Назначение
<i>Abbreviation</i>	Сокращенное имя валюты (первичный ключ);
<i>Name</i>	Полное имя валюты;
<i>IconUrl</i>	Ссылка на иконку для данной валюты.

При реализации данной модели было создано три поля в таблице «*Currency*», в которой в качестве первичного ключа выступает «*Abbreviation*».

2.2.5 Модель *TransactionType*

Данная модель содержит в себе типы транзакций, которые будут доступны пользователю для выбора при создании категорий для транзакций. Поля таблицы описаны в таблице 2.5.

Таблица 2.5 – Модель *TransactionType*

Поле	Назначение
<i>Name</i>	Полное имя типа (первичный ключ);
<i>IconUrl</i>	Ссылка на иконку для данного типа.

При реализации данной модели было создано два поля в таблице «*TransactionType*», в которой в качестве первичного ключа выступает «*Name*».

2.2.6 Модель BugReport

Данная модель содержит в себе отзывы пользователей, которые они могут оставлять на главной странице. Данная модель будет создана только в организации *Salesforce*, т.к. запрос на создание отзыва будет отправляться сразу непосредственно через *Salesforce API* и будет храниться только там, а не в базе данных *MSSQL*. Поля таблицы описаны в таблице 2.6.

Таблица 2.6 – Модель *BugReport*

Поле	Назначение
<i>Id</i>	Идентификатор отзыва (первичный ключ);
<i>Name</i>	Имя пользователя, оставившего отзыв;
<i>Email</i>	Электронная почта пользователя, оставившего отзыв;
<i>Phone</i>	Телефон пользователя, оставившего отзыв;
<i>Subject</i>	Заголовок проблемы или отзыва пользователя;
<i>Description</i>	Описание проблемы или полный отзыв пользователя.

При реализации данной модели было создано шесть полей в таблице «*BugReport*», в которой в качестве первичного ключа выступает «*Id*».

2.2.7 Модель UserEmailVerificationCode

Данная модель содержит в себе коды, которые были отправлены для подтверждения личности при операции восстановления доступа к аккаунту. Поля таблицы описаны в таблице 2.7.

Таблица 2.7 – Модель *UserEmailVerificationCode*

Поле	Назначение
<i>Id</i>	Идентификатор кода (первичный ключ);
<i>Code</i>	7-значный код из цифр, хранится строкой;
<i>ExpiredDate</i>	Дата, до наступления которой, этот код действительный;
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано четыре поля в таблице «*UserEmailVerificationCode*», в которой в качестве первичного ключа выступает «*Id*».

2.2.8 Модель UserRefreshToken

Данная модель содержит в себе *JWT*-токен, которые служат для обновления токенов доступа пользователя. Поля таблицы описаны в таблице 2.8.

Таблица 2.8 – Модель *UserRefreshToken*

Поле	Назначение
<i>Id</i>	Идентификатор токена (первичный ключ);
<i>Token</i>	<i>JWT</i> -токен, хранится строкой;
<i>ExpiredDate</i>	Дата, до наступления которой, этот токен остается валидным;
<i>Type</i>	Тип <i>JWT</i> -токена (токен приложения, токен пользователя), хранится как перечисление;
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано пять полей в таблице «*UserRefreshToken*», в которой в качестве первичного ключа выступает «*Id*».

2.2.9 Модель UserSetting

Данная модель содержит в себе настройки пользователей. Поля таблицы описаны в таблице 2.9.

Таблица 2.9 – Модель *UserSetting*

Поле	Назначение
<i>Id</i>	Идентификатор настройки (первичный ключ);
<i>Value</i>	Значение настройки пользователя, строка;
<i>SettingId</i>	Ссылка на настройку (внешний ключ);
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано четыре поля в таблице «*UserSetting*», в которой в качестве первичного ключа выступает «*Id*».

2.2.10 Модель UserNotification

Данная модель содержит в себе уведомления пользователей. Поля таблицы описаны в таблице 2.10.

Таблица 2.10 – Модель *UserNotification*

Поле	Назначение
<i>Id</i>	Идентификатор уведомления (первичный ключ);
<i>Title</i>	Заголовок уведомления;
<i>Content</i>	Содержимое уведомления;
<i>IsRead</i>	Флаг, показывающий, прочитано уведомление или нет;
<i>CreatedDate</i>	Дата создания уведомления;
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано шесть полей в таблице «*UserNotification*», в которой в качестве первичного ключа выступает «*Id*».

2.2.11 Модель Category

Данная модель содержит в себе категории транзакций. В приложении предусмотрен ряд стандартных категорий, однако пользователи могут создавать свои собственные категории, под их нужды. Поля таблицы описаны в таблице 2.11.

Таблица 2.11 – Модель *Category*

Поле	Назначение
<i>Id</i>	Идентификатор категории (первичный ключ);
<i>Name</i>	Имя категории;
<i>TypeName</i>	Ссылка на тип транзакции (внешний ключ);
<i>IsCustom</i>	Флаг, показывающий, является ли категория стандартной или нет;
<i>IconUrl</i>	Ссылка на иконку категории;
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано пять полей в таблице «*Category*», в которой в качестве первичного ключа выступает «*Id*».

2.2.12 Модель CurrencyExchangeRate

Данная модель содержит в себе курсы обмена валют. Пользователи могут сами добавлять курс, который они считают нужным, однако в приложении могут быть заранее добавленные курсы валют. Поля таблицы описаны в таблице 2.12.

Таблица 2.12 – Модель *CurrencyExchangeRate*

Поле	Назначение
<i>Id</i>	Идентификатор курса (первичный ключ);
<i>SourceCurrencyId</i>	Ссылка на изначальную валюту (внешний ключ);
<i>TargetCurrencyId</i>	Ссылка на получаемую валюту (внешний ключ);
<i>ValidFrom</i>	Дата, показывающая с какого дня этот курс валиден;
<i>ValidTo</i>	Дата, показывающая до какого дня этот курс валиден;
<i>ExchangeRate</i>	Курс обмена валют;
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано семь полей в таблице «*CurrencyExchangeRate*», в которой в качестве первичного ключа выступает «*Id*».

2.2.13 Модель Wallet

Данная модель содержит в себе кошельки пользователей. Пользователи могут сами создавать себе кошельки. При регистрации для каждого пользователя создается стандартный кошелек в валюте *USD*. Поля таблицы описаны ниже в таблице 2.13.

Таблица 2.13 – Модель *Wallet*

Поле	Назначение
<i>Id</i>	Идентификатор кошелька (первичный ключ);
<i>Name</i>	Имя кошелька;
<i>Balance</i>	Баланс кошелька;
<i>IsDefault</i>	Дата, показывающая до какого дня этот курс валиден;
<i>CurrencyAbbreviation</i>	Ссылка на валюту (внешний ключ);
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано шесть полей в таблице «*Wallet*», в которой в качестве первичного ключа выступает «*Id*».

2.2.14 Модель *Transaction*

Данная модель содержит в себе транзакции пользователей. Поля таблицы описаны в таблице 2.14.

Таблица 2.14 – Модель *Transaction*

Поле	Назначение
<i>Id</i>	Идентификатор транзакции (первичный ключ);
<i>Title</i>	Заголовок транзакции;
<i>Amount</i>	Сумма транзакции;
<i>Date</i>	Дата транзакции;
<i>CategoryId</i>	Ссылка на категорию (внешний ключ);
<i>WalletId</i>	Ссылка на кошелек (внешний ключ);
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано семь полей в таблице «*Transaction*», в которой в качестве первичного ключа выступает «*Id*».

2.2.15 Модель *RegularTransaction*

Данная модель содержит в себе повторяющиеся транзакции пользователей. Поля таблицы описаны в таблице 2.15.

Таблица 2.15 – Модель *RegularTransaction*

Поле	Назначение
<i>Id</i>	Идентификатор транзакции (первичный ключ);
<i>Title</i>	Заголовок транзакции;
<i>Amount</i>	Сумма транзакции;
<i>Date</i>	Дата транзакции;
<i>NextTransactionDate</i>	Дата следующей транзакции;
<i>Interval</i>	Интервал повторяющейся транзакции (день, неделя, месяц, год);
<i>CategoryId</i>	Ссылка на категорию (внешний ключ);

Окончание таблицы 2.15

Поле	Назначение
<i>WalletId</i>	Ссылка на кошелек (внешний ключ);
<i>UserId</i>	Ссылка на пользователя (внешний ключ).

При реализации данной модели было создано девять полей в таблице «*RegularTransaction*», в которой в качестве первичного ключа выступает «*Id*».

2.2.16 Контекст *CashSchedulerContext*

После создания классов моделей данных, необходимо создать контекст. Контекст – это специальный класс, предназначенный для создания запросов, отслеживания изменений и сохранения данных в базе. В рамках выполнения данного дипломного проекта был создан контекст *CashSchedulerContext*. Отображение его класса можно посмотреть в таблице 2.16.

Таблица 2.16 – Контекст *CashSchedulerContext*

Поле	Назначение
<i>DbSet<User> Users</i>	Создание контекста для модели <i>User</i> ;
<i>DbSet<UserRefreshToken> UserRefreshTokens</i>	Создание контекста для модели <i>UserRefreshToken</i> ;
<i>DbSet<UserEmailVerificationCode> UserEmailVerificationCodes</i>	Создание контекста для модели <i>UserEmailVerificationCode</i> ;
<i>DbSet<UserNotification> UserNotifications</i>	Создание контекста для модели <i>UserNotification</i> ;
<i>DbSet<UserSetting> UserSettings</i>	Создание контекста для модели <i>UserSetting</i> ;
<i>DbSet<Setting> Settings</i>	Создание контекста для модели <i>Setting</i> .
<i>DbSet<Language> Languages</i>	Создание контекста для модели <i>Language</i> ;
<i>DbSet<TransactionType> TransactionTypes</i>	Создание контекста для модели <i>TransactionType</i> ;
<i>DbSet<Category> Categories</i>	Создание контекста для модели <i>Category</i> ;
<i>DbSet<Transaction> Transactions</i>	Создание контекста для модели <i>Transaction</i> ;
<i>DbSet<RegularTransaction> RegularTransactions</i>	Создание контекста для модели <i>RegularTransaction</i> .
<i>DbSet<Wallet> Wallets</i>	Создание контекста для модели <i>Wallet</i> .
<i>DbSet<Currency> Currencies</i>	Создание контекста для модели <i>Currency</i> .
<i>DbSet<CurrencyExchangeRate> CurrencyExchangeRates</i>	Создание контекста для модели <i>CurrencyExchangeRate</i> .

Таким образом, рассмотренные выше классы моделей полностью описывают все данные, которыми будет оперировать система. Далее на их основе будут созданы таблицы в базе данных.

2.3 Структура приложения

Для будущей реализации веб-приложения необходимо определить его конкретную структуру. Таким образом оно будет более масштабируемым и понятным для изучения.

Данное веб-приложение разделяется на три главные части: клиентскую, серверную и организацию Salesforce. Каждая часть проекта является по сути независимым приложением и имеет отдельную директорию со своей структурой папок и файлов.

Структура серверной части приведена на рисунке 2.1.

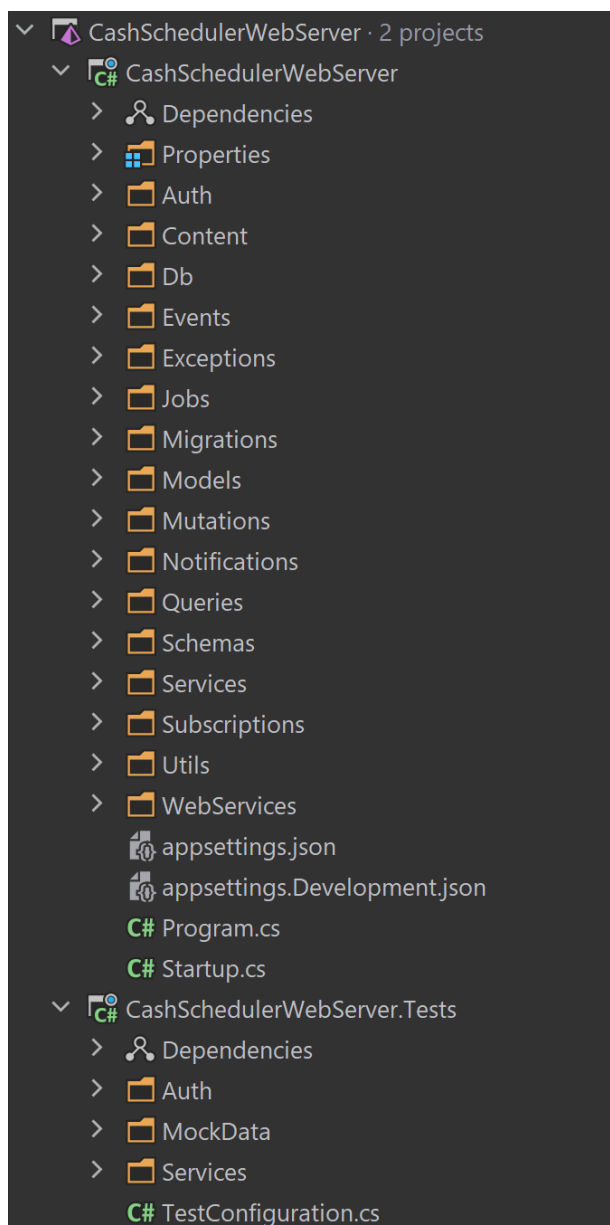


Рисунок 2.1 – Структура серверной части проекта

Здесь можно заметить отдельную папку с тестами, все тесты разделены в свою очередь по другим папкам в зависимости от того, за какую часть

функционала они отвечают. Кроме всего прочего в проекте есть разделение на модели, сервисы, веб-сервисы, классы для работы с базой данных и т.д.

Структура проекта для клиентской части представлена на рисунке 2.2.

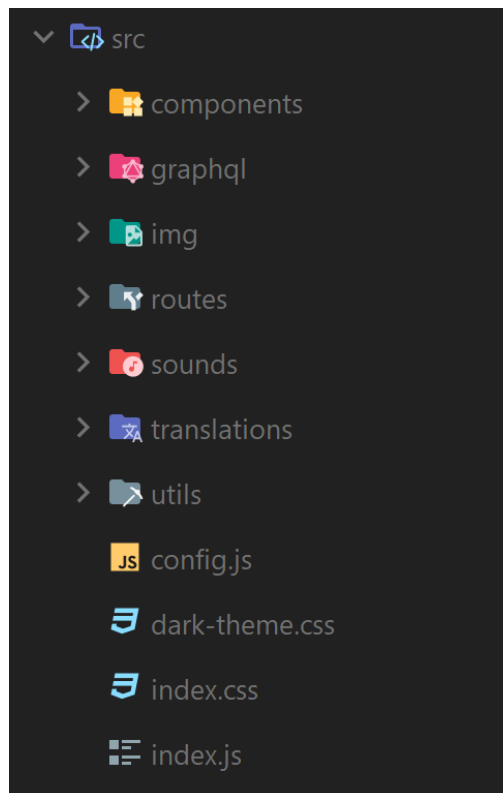


Рисунок 2.2 – Структура клиентской части проекта

Из важных моментов здесь можно отметить папку с компонентами, которая содержит в себе все *React*-компоненты приложения, папку «*graphql*», содержащую все *Query*, *Mutation* и *Subscription* запросы. Также есть папка с маршрутами и переводами, которые необходимы для возможности смены языка интерфейса в настройках.

Ну и последней частью проекта является *Salesforce* организация. *Salesforce* имеет множество настроек внешнего вида и работы самой платформы, поэтому структура проекта может выглядеть слегка перегруженной, однако это нормально. Частичная структура *Salesforce* проекта представлена на рисунке 2.3.

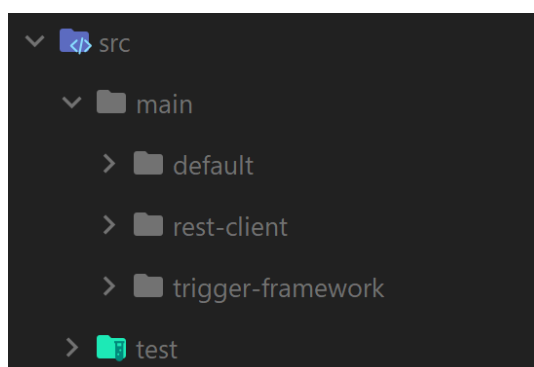


Рисунок 2.3 – Структура *Salesforce* проекта

Ввиду того, что проект содержит в себе десятки вложенных папок, здесь они все не показаны. Однако на рисунке выше можно увидеть, что исходный код и ресурсы находятся в папке «*main*» в то время, как тесты находятся в папке «*test*». В папке «*main*» также находятся и две Salesforce библиотеки, одна, для более удобного общения с *API* в *REST* стиле, другая для более удобного делегирования работы с триггерами в *Salesforce*. Остальные же классы, компоненты и ресурсы, разрабатываемые для данного веб-приложения находятся в папке «*default*».

Таким образом, подготовив оптимальную структуру каждой части проекта можно быть уверенным, что процесс разработки будет намного более плодотворным и в конечном итоге веб-приложение получится более качественным и масштабируемым.

2.4 Вывод по разделу

В результате проектирования и анализа поставленной задачи для реализации дипломного проекта были выбраны платформы *ASP.NET Core* и *Salesforce*, так как они имеют ряд преимуществ, описанных в первом разделе.

Для разработки базы данных используется фреймворк *Entity Framework*, в качестве взаимодействия используется *code first*.

В качестве системы управления базой данных выбор был сделан в пользу *Microsoft SQL Server*.

После создания всех необходимых моделей и контекста базы данных, при первом запуске приложения сгенерирована база данных. Логическая схема базы данных представлена в приложении Б.

Благодаря использованию *Entity Framework*, все таблицы были созданы без единой строчки *SQL*-кода, и они абсолютно идентичны классам моделей.

Разработанные модели данных подробно описаны в разделе «Описание моделей базы данных».

Для разработки клиентской части будет использована библиотека *React JS*. Серверная часть будет работать на платформе *ASP.NET Core* и общаться с платформой *Salesforce*.

Программный веб-интерфейс будет разработан с помощью языка запросов *GraphQL* на базе пакета *Hot Chocolate*, описанных в первом разделе.

3 Разработка веб-приложения

3.1 Описание разработанных GraphQL резолверов

Резолвер – это специальный метод или класс (в зависимости от реализации *GraphQL* пакета), который отвечает за обработку вызова *GraphQL* метода или поля. Если в рамках запросов необходимо получить какие-то данные, контроллер обращается в базу данных через контекст.

В рамках выполнения данного дипломного проекта были разработаны резолверы, которые рассмотрены ниже.

3.1.1 Резолверы модели User

Резолверы модели *User* представлены как в виде *Query*, так и в виде *Mutation*.

Query резолверы отвечают за следующие функции:

- получение информации о текущем пользователе;
- получения токена для доступа сторонних приложений;
- отправка и подтверждение доступа к аккаунту через *email*.

Mutation резолверы отвечают за следующие функции:

- регистрация и вход/выход пользователя из системы;
- генерация токена доступа;
- обновление пароля пользователя;
- обновление информации пользователя;
- удаление пользователя.

Список резолверов, их параметров и возвращаемых типов представлен в листингах 3.1, 3.2 и 3.3.

```
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public User User([Service] IContextProvider contextProvider)
{...}

[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new []{AuthOptions.USER_ROLE})]
public Task<string> AppToken([Service] IAuthService authService)
{...}

public Task<string> CheckEmail([Service] IAuthService authService,
[GraphQLNonNullType] string email)
{...}
```

					БГТУ 03.00.ПЗ								
Изм	Лист	№ докум.	Подп.	Дата									
Разраб.		Мацуев И.М.			3 Разработка веб-приложения				Лит.	Лист	Листов		
Пров.		Северинчик Н.А.									1	25	
Консульт.		Северинчик Н.А.							74417006, 2021				
Н. контр.		Рыжанкова А.С.											
Утв.		Пацей Н.В.											

```

public Task<string> CheckCode(
    [Service] IAuthService authService,
    [GraphQLNonNullType] string email,
    [GraphQLNonNullType] string code)
{...}

```

Листинг 3.1 – *Query* резолверы модели *User*

Метод *User* отвечает за получение информации о пользователе. Он помечен атрибутом *Authorize*, который гарантирует, что доступ к этому резолверу будет доступен только запросам, которые соответствуют заранее заданной политике *Policy*. Принимает в качестве параметра (инъекции зависимости) проводник контекста, через который можно получить доступ ко всем сервисам и репозиториям приложения. Метод возвращает модель *User*, по токену доступа, который был получен вместе с запросом.

Метод *AppToken* возвращает токен доступа с ролью приложения. Это специальный токен, который ограничен в некоторых запросах и может быть использован сторонними сервисами, для автоматического создания данных под учетной записью пользователя. Кроме атрибута *Authorize* с политикой, он также помечен параметром *Roles*, который позволяет вызывать этот метод только пользователям с пользовательским токеном доступа, чтобы никто кроме пользователя не мог получить этот токен доступа. В качестве инъекции принимает сервис *AuthService*, который предоставляет базовые методы для индентификации, аутентификации и авторизации. Метод возвращает строку – токен приложения.

Метод *CheckEmail* отправляет письмо с кодом на почту, переданную в параметре метода, как обязательный. Код имеет время жизни и сохраняется в базе данных. Метод возвращает почту, переданную в параметре в качестве заглушки.

Метод *CheckCode* нужен для проверки подленности кода, который был передан в параметрах с кодом, который был отправлен на почту. В случае несоответствия кодов или не имени пользователей в системе с почтой, переданной в параметрах, будет выброшено исключение. В случае успеха, возвращается почта пользователя в качестве заглушки.

```

[GraphQLNonNullType]
public Task<User> Register([Service] IAuthService authService,
[GraphQLNonNullType] NewUserInput user)
{...}
[GraphQLNonNullType]
public Task<AuthTokens> Login(
    [Service] IAuthService authService,
    [GraphQLNonNullType] string email,
    [GraphQLNonNullType] string password)
{...}

[GraphQLNonNullType]
public Task<AuthTokens> AppLogin([Service] IAuthService authService,
[GraphQLNonNullType] string appToken)
{...}

```

```

    [GraphQLNonNullType]
    [Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
    public Task<User> Logout([Service] IAuthService authService)
    {...}

    [GraphQLNonNullType]
    [Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
    public Task<User> LogoutConnectedApps([Service] IAuthService authService)
    {...}

    [GraphQLNonNullType]
    public Task<AuthTokens> Token(
        [Service] IAuthService authService,
        [GraphQLNonNullType] string email,
        [GraphQLNonNullType] string refreshToken)
    {...}

```

Листинг 3.2 – *Mutation* резолверы модели *User*

Метод *Register* отвечает за создание нового пользователя. Принимает в качестве параметра *Input* модель юзера, в которую входит имя, фамилия, баланс, почта и пароль пользователя. В зависимости от этих данных создается новый пользователь и возвращается как результат.

Метод *Login* нужен для входа пользователя в приложение. С технической точки зрения, он нужен для получения двух токенов: токена доступа и токена обновления. Первый нужен для доступа пользователя к методам специфическим для пользователя (например, создание транзакции, обновление и т.д.). По почте и паролю, передаваемым как параметры, находится пользователь в БД и возвращается. Пароль хранится в захешированном виде.

AppLogin похож на обычный метод *Login* тем, что также возвращает два токена, но сам метод и токены, которые он возвращает предназначены только для сторонних приложений, которым пользователь захотел дать доступ. Этим методом будет пользоваться какой-либо сторонний сервис, передавая в качестве параметра токен доступа приложения и получая два токена, с ролью внешнего приложения.

Метод *Logout* позволяет пользователю выйти из приложения и удалить всю информацию о токенах обновления, которые удаляются из базы данных. Таким образом, никто, у кого до этого был токен обновления, не сможет войти в приложение по нему – придется использовать логин и пароль.

LogoutConnectedApps так же удаляет все токены обновления из базы данных, но на этот раз это все токены с ролью приложения, что позволяет ограничить доступ к данным пользователя все приложениям, которым был предоставлен доступ доступа.

Метод *Token* нужен для генерации двух новых токенов, но на этот не по почте и паролю, а по почте и токену обновления, которые могли быть сгенерированы до этого с помощью метода *Login*.

```
[GraphQLNonNullType]
public Task<User> ResetPassword(
    [Service] IAuthService authService,
    [GraphQLNonNullType] string email,
    [GraphQLNonNullType] string code,
    [GraphQLNonNullType] string password)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<User> UpdateUser(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] UpdateUserInput user)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<User> DeleteUser(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] string password)
{...}
```

Листинг 3.3 – *Mutation* резолверы модели *User*

ResetPassword позволяет поменять пароль пользователя, проверяя входящие код и почту пользователя, которые были подтверждены путем отправки кода на почту. Новый пароль также предоставляется в виде параметра.

Метод *UpdateUser* нужен для обновления информации пользователя, по сути менять он может только имя, фамилию и баланс своего кошелька по умолчанию. В параметрах передается *Input* с соответствующими полями.

Метод *DeleteUser* позволяет пользователю полностью удалить его аккаунт и все личные данные. Это защищенный метод, который может выполнить только пользователь, подтвердив свой пароль. При выполнении метода, в очередь становится удаление всех данных пользователя с задержкой в 1 час. За это время пользователь может сделать последние действия со своим аккаунтом, сохранить какую-либо информацию или что-то еще. После этого пользователь будет полностью удален из базы данных.

3.1.2 Резолверы модели *Category*

Эти резолверы служат для обработки запросов связанных с созданием, обновлением, удалением и получением категорий.

Query резолверы реализуют следующие функции:

- получение стандартных и пользовательских категорий сразу;
- получение стандартных категорий;
- получение пользовательских категорий.

Mutation резолверы реализуют следующие функции:

- создание категории;
- обновление категории;
- удаление категории.

Список резолверов, их параметров и возвращаемых типов представлен в листингах 3.4 и 3.5.

```
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<Category>? AllCategories([Service] IContextPro-
vider contextProvider, string? transactionType)
{...}

public IEnumerable<Category> StandardCategories(
    [Service] IContextProvider contextProvider,
    string? transactionType)
{...}

[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<Category>? CustomCategories(
    [Service] IContextProvider contextProvider,
    string? transactionType)
{...}
```

Листинг 3.4 – *Query* резолверы модели *Category*

Метод *AllCategories* предназначен для возвращений всех категорий, к которым у пользователя есть доступ, в том числе к стандартным. Также, есть необязательный параметр *transactionType*, который можно задать, чтобы получить категории определенного типа – дохода или расхода.

Метод *StandardCategories* предназначен для возвращения только стандартных категорий. Этот метод можно выполнять даже не авторизованным пользователям.

Метод *CustomCategories* предназначен для возвращения только пользовательских категорий.

```
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Category> CreateCategory(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewCategoryInput category)
{...}
```



```

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Category> CreateCategory(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewCategoryInput category)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Category> UpdateCategory(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] UpdateCategoryInput category)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<Category> DeleteCategory([Service] IContextProvider contextProvider, [GraphQLNonNullType] int id)
{...}

```

Листинг 3.5 – *Mutation* резолверы модели *Category*

Метод *CreateCategory* нужен для создания новых пользовательских категорий. Принимает обязательный *Input* параметр с такими полями как имя категории, тип и ссылка на иконку, если таковая имеется. Если ссылка на иконку не была предоставлена, ей будет задана стандартная иконка. Возвращает созданную категорию.

Метод *UpdateCategory* нужен для обновления пользовательских категорий. Пользователи не могут обновлять или удалять стандартные категории. Обновить у категории можно только имя и ссылку на иконку. Возвращает обновленную категорию.

Метод *DeleteCategory* предназначен для удаления пользовательских категорий. В качестве параметра принимает идентификатор нужной категории. Метод доступен только роли пользователя. Сторонние приложения не смогут удалять категории из соображений безопасности. При удалении категории, все связанные с ней транзакции будут также удалены. Возвращает удаленную категорию.

3.1.3 Резолверы модели *Wallet*

Эти резолверы необходимы для манипуляции с кошельками пользователя. *Query* резолверы реализуют следующие функции:

- получение всех кошельков пользователя;
- получение баланса кошелька по умолчанию.

Mutation резолверы реализуют следующие функции:

- создание/обновление/удаление кошелька;

– создание перевода между кошельками.

Список резолверов, их параметров и возвращаемых типов представлен в листингах 3.6 и 3.7.

```
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<Wallet>? Wallets([Service] IContextProvider contextProvider)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public double Balance([Service] IContextProvider contextProvider)
{...}
```

Листинг 3.6 – *Query* резолверы модели *Wallet*

Метод *Wallets* предназначен для возвращения всех кошельков пользователя.

Метод *Balance* предназначен для возвращения значения баланса кошелька пользователя по умолчанию. У пользователя всегда есть кошелек по умолчанию. Его можно сменить, но его нельзя убрать.

```
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Wallet> CreateWallet(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewWalletInput wallet) {...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Wallet> UpdateWallet(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] UpdateWalletInput wallet) {...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<Wallet> DeleteWallet([Service] IContextProvider contextProvider, [GraphQLNonNullType] int id) {...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<Transfer> CreateTransfer(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewTransferInput transfer) {...}
```

Листинг 3.7 – *Mutation* резолверы модели *Wallet*

Метод *CreateWallet* предназначен для создания кошелька. В параметрах передается *Input* объект с такими полями как имя кошелька, баланс, валюта и флаг, показывающий, будет ли кошелек выбран по умолчанию.

Метод *UpdateWallet* предназначен для обновления кошелька. Обновить у кошелька можно любое поле. При смене валюты кошелька, можно также указать, что вы хотите пересчитать баланс согласно указанного курса из одной валюты в другую.

Метод *DeleteWallet* предназначен для удаления кошелька. Если кошелек выбран по умолчанию, то его нельзя удалить. В качестве параметра передается идентификатор кошелька. При удалении кошелька все транзакции, ссылающиеся на него, будут также удалены.

Метод *CreateTransfer* нужен для перевода суммы с одного кошелька на другой. При создании перевода нужно указать кошельки, откуда и куда будем переводить, сумму, а также курс, по которому будет совершаться перевод. После выполнения метода, балансы двух кошельков будут обновлены.

3.1.4 Резолверы модели *Transaction*

Эти резолверы необходимы для манипуляции с транзакциями пользователя. Создание, обновление и удаление транзакций сказывается на балансе кошельков, к которым они привязаны.

Query резолверы реализуют следующие функции:

- получение транзакций для отображения на главной странице;
- получение транзакций за конкретный месяц;
- получение разницы суммы транзакций помесечно за конкретный год.

Mutation резолверы реализуют следующие функции:

- создание/обновление/удаление транзакций.

Список резолверов, их параметров и возвращаемых типов представлены в листингах 3.8 и 3.9.

```
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<Transaction>? DashboardTransactions(
    [Service] IContextProvider contextProvider,
    int month, int year) {...}

[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<Transaction>? TransactionsByMonth(
    [Service] IContextProvider contextProvider,
    int month,
    int year) {...}

[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public IEnumerable<TransactionDelta>? TransactionsDelta([Service]
IContextProvider contextProvider, int year, bool isRecurring = false) {...}
```

Листинг 3.8 – *Query* резолверы модели *Transaction*

Метод *DashboardTransactions* предназначен для получения транзакций для отображения на главной странице приложения. По сути этот метод просто возвращает транзакции за определенный месяц, плюс, транзакции за месяц до этого и за месяц после этого. Это необходимо для корректного отображения информации, т.к. главная страница спроектирована в виде календаря, где одновременно на одной странице может быть отображено 3 месяца (конец предыдущего, текущий месяц и начало следующего). Принимает в качестве параметра месяц и год.

Метод *TransactionsByMonth* предназначен для получения транзакций за один конкретный месяц, который передается в качестве параметра вместе с годом.

Метод *TransactionsDelta* предназначен для получения сводки суммы транзакций за каждый месяц в течении года, переданного в параметрах. Этот метод возвращает массив из wrappers с двумя поля: месяц и сумма разницы. Этот метод предназначен для отображения одного из графиков на странице транзакций.

```
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Transaction> CreateTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewTransactionInput transaction)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<Transaction> UpdateTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] UpdateTransactionInput transaction)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<Transaction> DeleteTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] int id)
{...}
```

Листинг 3.9 – *Mutation* резолверы модели *Transaction*

Метод *CreateTransaction* предназначен для создания транзакции. В качестве параметра передается *Input* с такими полями как заголовок транзакции, идентификатор категории и кошелька, сумма транзакции и дата.

Метод *UpdateTransaction* предназначен для обновления транзакции. Обновлять у транзакций можно заголовок, сумму и дату.

Метод *DeleteTransaction* предназначен для удаления транзакции. Передается идентификатор транзакции в качестве параметра.

3.1.5 Резолверы модели *RegularTransaction*

Эти резолверы необходимы для манипуляции с повторяющимися транзакциями пользователя. Создание, обновление и удаление повторяющихся транзакций не влияет на баланс кошельков, к которым они привязаны. Однако, по информации регулярных транзакций, в зависимости от даты и интервала, создаются обычные транзакции, резолверы которых были описаны в подразделе выше.

Query резолверы реализуют следующие функции:

- получение повторяющихся транзакций для отображения на главной странице;

- получение повторяющихся транзакций за конкретный месяц.

Mutation резолверы реализуют следующие функции:

- создание/обновление/удаление повторяющихся транзакций.

Список резолверов, их параметров и возвращаемых типов представлен в листингах 3.10 и 3.11.

```
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<RegularTransaction>? DashboardRecurringTransactions(
    [Service] IContextProvider contextProvider,
    int month,
    int year)
{...}

[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public IEnumerable<RegularTransaction>? RecurringTransactionsByMonth(
    [Service] IContextProvider contextProvider,
    int month,
    int year)
{...}
```

Листинг 3.10 – *Query* резолверы модели *RegularTransaction*

Метод *DashboardRecurringTransactions* предназначен для получения повторяющихся транзакций для отображения на главной странице приложения. Работает по такой же логике, как и аналогичный резолвер для модели *Transaction*, только нужен для модели *RegularTransaction*.

Метод *TransactionsByMonth* предназначен для получения повторяющихся транзакций за один конкретный месяц, который передается в качестве параметра вместе с годом.

```
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<RegularTransaction> CreateRegularTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewRecurringTransactionInput transaction)
```

```

{...}
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<RegularTransaction> CreateRegularTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewRecurringTransactionInput transaction)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY)]
public Task<RegularTransaction> UpdateRegularTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] UpdateRecurringTransactionInput transac-
tion)
{...}

[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOp-
tions.USER_ROLE})]
public Task<RegularTransaction> DeleteRegularTransaction(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] int id)
{...}

```

Листинг 3.11 – *Mutation* резолверы модели *RegularTransaction*

Метод *CreateRegularTransaction* предназначен для создания повторяющейся транзакции. В качестве параметра передается *Input* с такими полями как заголовок транзакции, идентификатор категории и кошелька, сумма транзакции, интервал и дата следующего срабатывания транзакции.

Метод *UpdateRegularTransaction* предназначен для обновления повторяющейся транзакции. Обновлять у повторяющихся транзакций можно только заголовок и сумму.

Метод *DeleteRegularTransaction* предназначен для удаления повторяющейся транзакции. Передается идентификатор транзакции в качестве параметра.

3.1.6 Резолверы модели *UserSetting*

Эти резолверы необходимы для получения и изменения настроек пользователя.

Query резолверы реализуют следующие функции:

- Получение имен всех настроек;
- Получение имен всех разделов настроек;
- Получение имен всех секций настроек;
- Получение всех возможных для выбора языков;
- Получение текущих настроек пользователя.

Mutation резолверы реализуют следующие функции:

- Обновления одной или нескольких настроек пользователя.

Список резолверов, их параметров и возвращаемых типов представлен в листингах 3.12 и 3.13.

```
[GraphQLNonNullType]
public IEnumerable<string> SettingNames()
{...}

[GraphQLNonNullType]
public IEnumerable<string> SettingUnits()
{...}

[GraphQLNonNullType]
public IEnumerable<string> SettingSections()
{...}

[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public IEnumerable<UserSetting>? Settings([Service] IContextProvider
contextProvider, string? unitName)
{...}

public IEnumerable<Language>? Languages([Service] IContextProvider
contextProvider)
{...}
```

Листинг 3.12 – *Query* резолверы модели *UserSetting*

Метод *SettingNames* возвращает список имен всех возможных настроек как список строк.

Метод *SettingUnits* возвращает список имен всех возможных разделов настроек как список строк.

Метод *SettingSections* возвращает список имен всех возможных секций настроек как список строк.

Метод *Languages* возвращает список всех возможных для выбора языков. Возвращает список модели *Language*.

Метод *Settings* возвращает список всех настроек пользователя в зависимости от имени раздела, передаваемого как параметр.

```
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<UserSetting> UpdateUserSetting(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] UpdateUserSettingInput setting) {...}

[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
```

```

public Task<IEnumerable<UserSetting>?> UpdateUserSettings(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] IEnumerable<UpdateUserSettingInput>
        Settings
    ) {...}

```

Листинг 3.13 – *Mutation* резолверы модели *UserSetting*

Метод *UpdateUserSetting* предназначен для обновления настройки пользователя. В качестве параметра передается *Input* с такими полями как имя настройки и ее новое значение.

Метод *UpdateUserSettings* полностью аналогичен методу *UpdateUserSetting* за тем исключением, что этот резолвер предназначен для обновления множества настроек за раз. В качестве параметра передается список *Input*'а, описанного выше.

3.1.7 Резолверы модели *UserNotification*

Эти резолверы необходимы для получения и изменения уведомлений пользователя.

Query резолверы реализуют следующие функции:

- получение всех уведомлений пользователя;
- получение количества непрочитанных уведомлений.

Mutation резолверы реализуют следующие функции:

- отметка уведомления, как прочитанного или непрочитанного;
- создание уведомления.

Subscription резолверы реализуют следующие функции:

- прослушка созданных уведомлений.

Список резолверов, их параметров и возвращаемых типов представлен в листингах 3.14, 3.15 и 3.16.

```

    [Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
    public IEnumerable<UserNotification>? Notifications([Service] IContextProvider contextProvider)
    {...}

    [GraphQLNonNullType]
    [Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
    public int UnreadNotificationsCount([Service] IContextProvider contextProvider)
    {...}

```

Листинг 3.14 – *Query* резолверы модели *UserNotification*

Метод *Notifications* возвращает список всех уведомлений пользователя.

Метод *UnreadNotificationsCount* возвращает число, определяющее количество непрочитанных сообщений у пользователя.

```
[GraphQLNonNullType]
[Authorize(Policy = AuthOptions.AUTH_POLICY, Roles = new[] {AuthOptions.USER_ROLE})]
public Task<UserNotification> ToggleReadNotification(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] int id,
    [GraphQLNonNullType] bool read)
{...}

[GraphQLNonNullType]
[Authorize(Roles = new[] {AuthOptions.SALESFORCE_ROLE})]
public Task<UserNotification> CreateNotification(
    [Service] IContextProvider contextProvider,
    [GraphQLNonNullType] NewNotificationInput notification)
{...}
```

Листинг 3.15 – *Mutation* резолверы модели *UserNotification*

Метод *ToggleReadNotification* предназначен для пометки уведомления как прочитанного или непрочитанного, в зависимости от параметра *read*. Также передается идентификатор нужного уведомления.

Метод *CreateNotification* предназначен для создания уведомления. Доступен этот резолвер только роли *Salesforce*, т.к. пользователь не может создать уведомление сам для себя.

```
[SubscribeAndResolve]
public async ValueTask<IStream<UserNotification>> OnNotificationCreated(
    [Service] ITopicEventReceiver eventReceiver,
    int userId,
    CancellationToken cancellationToken)
{...}
```

Листинг 3.16 – *Subscription* резолверы модели *UserNotification*

Метод *OnNotificationCreated* предназначен для уведомления подключенных клиентов, о том, что было создано новое уведомление. В качестве параметра принимается идентификатор пользователя, которому будут приходить уведомления. Также в качестве параметра здесь присутствует *CancellationToken*, который нужен для отмены запроса при прерывании соединения с клиентом. Для того, чтобы подписаться на нужный ивент здесь также делается инъекция объекта *ITopicEventReceiver*, с помощью которого можно подписаться на событие создания уведомления.

3.2 Настройка и установка библиотеки Hot Chocolate

Для начала необходимо в проект добавить саму библиотеку. Для этого необходимо зайти в *NuGet* менеджер и установить пакет *HotChocolate*.

После установки библиотеки, чтобы задействовать ее функциональность, необходимо настроить *DI* контейнер в классе *Startup*, в методе *ConfigureServices*. Код конфигурации пакета показан в листинге 3.17.

```
services.AddGraphQLServer()
    .AddQueryType<Query>()
        .AddTypeExtension<UserQueries>()
        .AddTypeExtension<TransactionTypeQueries>()
        .AddTypeExtension<CategoryQueries>()
        .AddTypeExtension<TransactionQueries>()
        .AddTypeExtension<RecurringTransactionQueries>()
        .AddTypeExtension<UserNotificationQueries>()
        .AddTypeExtension<UserSettingQueries>()
        .AddTypeExtension<WalletQueries>()
        .AddTypeExtension<CurrencyQueries>()
        .AddTypeExtension<CurrencyExchangeRateQueries>()
        .AddTypeExtension<SalesforceQueries>()
    .AddMutationType<Mutation>()
        .AddTypeExtension<UserMutations>()
        .AddTypeExtension<CategoryMutations>()
        .AddTypeExtension<TransactionMutations>()
        .AddTypeExtension<RecurringTransactionMutations>()
        .AddTypeExtension<NotificationMutations>()
        .AddTypeExtension<SettingMutations>()
        .AddTypeExtension<WalletMutations>()
        .AddTypeExtension<CurrencyExchangeRateMutations>()
        .AddTypeExtension<SalesforceMutations>()
    .AddSubscriptionType<Subscription>()
        .AddTypeExtension<UserNotificationSubscriptions>()
    .AddAuthorization()
    .AddInMemorySubscriptions()
    .AddErrorFilter<CashSchedulerErrorFilter>();
```

Листинг 3.17 – Конфигурация *HotChocolate*

Как видно на рисунке, для настройки *GraphQL* сервера используется метод *AddGraphQLServer*, который конфигурируется путем добавления типов и расширений с помощью методов *AddQueryType*, *AddMutationType*, *AddSubscriptionType* и *AddTypeExtension*. Метод *AddAuthorization* позволит нам использовать стандартную функциональность авторизации *ASP.NET Core* в виде атрибутов *Authorization* с использованием политик и ролей. Для работы *Subscriptions* нам необходимо добавить специальный провайдер, который будет определять, где хранится ответ, который будут получать клиенты. В нашем случае это *AddInMemorySubscriptions*. Для обработки ошибок добавляются фильтры, которые

вызываются например при ошибке структуры *GraphQL* запроса или когда мы просто бросаем исключение в резолверах. Тут можно корректировать ответ, который будут получать клиенты при выпадении ошибок.

Для применения нашей конфигурации и запуска сервера необходимо настроить метод *Configure* в *Startup.cs*. Окончательный вариант метода *Configure* представлен в листинге 3.18.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    CashSchedulerSeeder.InitializeDb(app, Configuration);
    app.UseStaticFiles(GetStaticFileOptions(Configuration));
    app.UseWebSockets();
    app.UseRouting();
    app.UseCors();
    app.UseAuthentication();
    app.UseAuthorization();
    app.UsePlayground(
        Configuration["App:Server:GraphQLAPIPath"],
        Configuration["App:Server:GraphQLPlaygroundPath"]
    );
    app.UseHttpsRedirection();
    app.UseEndpoints(
        endpoints => endpoints
            .MapGraphQL(
                Configuration["App:Server:GraphQLAPIPath"]
            )
    );
}
```

Листинг 3.18 – Метод *Configure*

На рисунке выше изображена последовательность выполнения функций — *middleware*, здесь настраивается миграция базы данных при первом запуске, использование статических файлов для иконок категорий, настройка веб сокетов, роутинг, настройка политики *CORS*, аутентификация, авторизация, песочница для *GraphQL* сервера для проверки работоспособности *API*, переадресация на *HTTPS* и сам по себе маппинг *URI*, на котором будет находиться наш сервер. После этого наш сервер готов к работе и отладке.

3.3 Реализация интеграции с *Salesforce*

В данном курсовом проекте *Salesforce* используется как место для сбора хранения статистики использования приложения, а также для службы поддержки клиентов. Для этого было необходимо, чтобы данные, создаваемые и изменяемые пользователями автоматически передавались в *Salesforce* организацию, где в конечном итоге будет происходить их обработка. Для этого была реализована

схема подписки и отправки ивентов, а также сервис, который будет общаться со стандартным *Salesforce REST API*. Все необходимые таблицы были реплецированы в *Salesforce* заранее.

3.3.1 Реализация схемы издатель-подписчик

Для реализации схемы издатель-подписчик были разработаны:

- перечисление *EventAction*, содержащий в себе все возможные события (представлен в листинге 3.19);

- интерфейс *IEventListener*, содержащий свойство типа *EventAction*, определяющее, за какое событие отвечает слушатель и лишь один метод, который будет являться подписчиком (представлен в листинге 3.20);

класс *EventManager*, необходимый для отправки события и вызова всех нужных подписчиков (представлен в листинге 3.21).

```
public enum EventAction
{
    UserLogin,
    UserRegistered,
    UserDeleted,
    RecordUpserted,
    RecordDeleted
}
```

Листинг 3.19 – Перечисление *EventAction*

EventAction имеет в себе события для оповещения о том, что пользователь вошел в систему, зарегистрировался, удалил свой аккаунт, либо о том, что какая-либо запись была создана, обновлена или удалена.

```
public interface IEventListener
{
    EventAction Action { get; }
    Task Handle(object entity);
}
```

Листинг 3.20 – Интерфейс *IEventListener*

Чтобы стать слушателем событий класс должен реализовать интерфейс *IEventListener*. Для этого необходимо задать значение событию, за которое он будет отвечать, а также реализовать метод *Handle*, чтобы определить, что будет происходить при возникновении события.

```
public class EventManager : IEventManager
{
    private IEnumerable<IEventListener> Listeners { get; }
```

```

public EventManager(IEnumerable<IEventListener> listeners)
{
    Listeners = listeners;
}

public async Task FireEvent(EventAction action, object entity)
{
    foreach (var listener in Listeners.Where(l => l.Action ==
action))
    {
        await listener.Handle(entity);
    }
}
}

```

Листинг 3.21 – Класс *EventManager*

Чтобы отправить событие, будет вызываться метод *FireEvent*, который принимает в себе сам тип события и параметр типа *object*, который может быть любым ссылочным типом. Метод просто находит всех слушателей с таким же типом события и вызывает каждый из них. Остается только зарегистрировать *EventManager* и все реализации слушателей в *DI* контейнер *ASP.NET Core* (представлен в листинге 3.22).

```

services.AddScoped<IEventManager, EventManager>();
services.AddScoped<IEventListener, CreateDefaultWalletListener>();
services.AddScoped<IEventListener, CreateDefaultSettingsListener>();
services.AddScoped<IEventListener, CreateSfContactListener>();
services.AddScoped<IEventListener, DeleteSfContactListener>();
services.AddScoped<IEventListener, UpsertSfRecordListener>();
services.AddScoped<IEventListener, DeleteSfRecordListener>();
services.AddScoped<IEventListener, LogUserLoginListener>();

```

Листинг 3.22 – Регистрация зависимостей для реализации событий

Теперь у нас есть возможность делать инъекции *IEventManager* и вызывать нужные нам события независимо основного потока программы.

3.3.2 Реализация сервиса для общения с Salesforce REST API

Salesforce имеет свой стандартный *REST* и *SOAP API* для работы со стандартными и пользовательскими объектами. Поэтому задача стояла только в написании сервиса для общения с этим *API* из моего *ASP.NET Core* приложения. Интерфейс сервиса представлен в листинге 3.23.

```

public interface ISalesforceApiWebService
{
    Task<string> Login();

    Task<string> Login(
        string clientId,
        string clientSecret,
        string username,
        string password,
        string securityToken
    );

    Task UpsertSObject(SfObject sObject);

    Task UpsertSObjects(List<SfObject> sObjects);

    Task DeleteSObject(SfObject sObject);

    Task DeleteSObjects(List<SfObject> sObjects);

    void RunWithDelay(
        SfObject sObject,
        int delay,
        Action<ISalesforceApiWebService, SfObject> action
    );

    Task CreateCase(SfCase sObject);
}

```

Листинг 3.23 – Интерфейс *Salesforce* сервиса

Метод *Login* необходим для получения токена доступа, с помощью которого можно выполнять операции с данными. Он принимает идентификатор клиента и секрет, которые настраиваются в *Salesforce* организации, а также юзернейм и пароль от пользователя в организации и токен безопасности, к которому имеет доступ только *Salesforce* администратор.

Метод *UpsertSObject* необходим для создания или обновления записей в организации, принимает в качестве аргумента wrapper *Salesforce* объекта, который может быть пользователем, категорией, транзакцией и т.д.

Метод *UpsertSObjects* аналогичен методу выше, но предназначен для множественного создания или обновления записей.

Методы *DeleteSObject* и *DeleteSObjects* нужны для удаления одной или множества записей в *Salesforce* организации.

Метод *RunWithDelay* позволяет запланировать отложенную операцию с записью на определенное количество секунд, благодаря делегату *Action*.

Метод *CreateCase* нужен для создания «кейса» (по сути отзыва или запроса в поддержку) в организации.

Каждый из описанных выше методов в своей реализации отправляет *POST*

HTTP запросы на стандартный *Salesforce endpoint*, специфический для каждой организации. Общение происходит в формате *JSON* и *URLENCODED*.

Таким образом, у нас теперь есть возможность отправлять запросы на создание, обновление и удаление на организацию *Salesforce*, что позволит нам все время иметь актуальные данные для работы с графиками и обработкой информации.

3.3.3 Настройка Salesforce организации

Как я написал в начале раздела, все модели в организации были заранее реплецированы и созданы с теми же полями, что и в *MSSQL* БД. После этого были настроены два приложения: приложения для общего просмотра информации и отчетов и приложение для центра поддержки.

Для первого приложения был подправлен внешний вид организации, а также организовано пространство для удобства просмотра информации (рисунок 3.1).

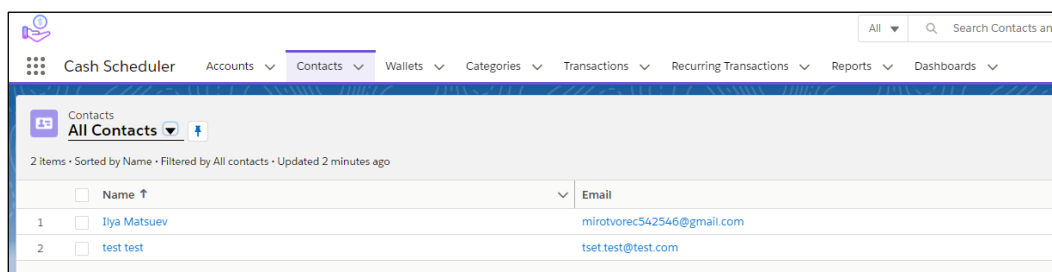


Рисунок 3.1 – Рабочее пространство *Salesforce* организации

Приложение было брэндировано и настроены вкладки для всех нужных моделей, а также добавлены вкладки с отчетами и графиками.

Страница записи пользователя показана на рисунке 3.2.

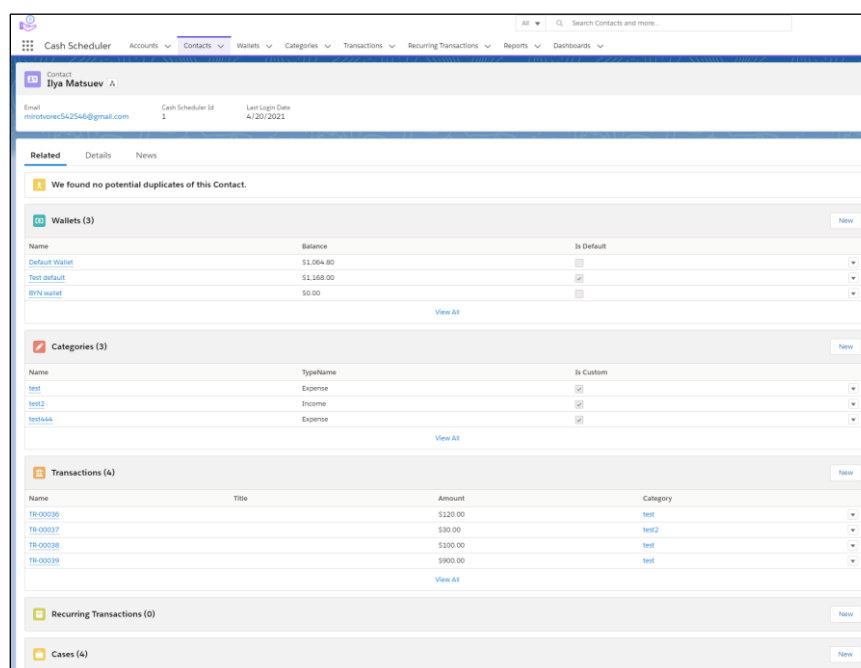


Рисунок 3.2 – Страница детального просмотра записи

Здесь можно удобно просматривать и переходить на все связанные записи. Аналогично и на других страницах моделей.

Помимо этого была добавлена вкладка с репортами, где было подготовлено несколько репортов, показанных на рисунке 3.3.

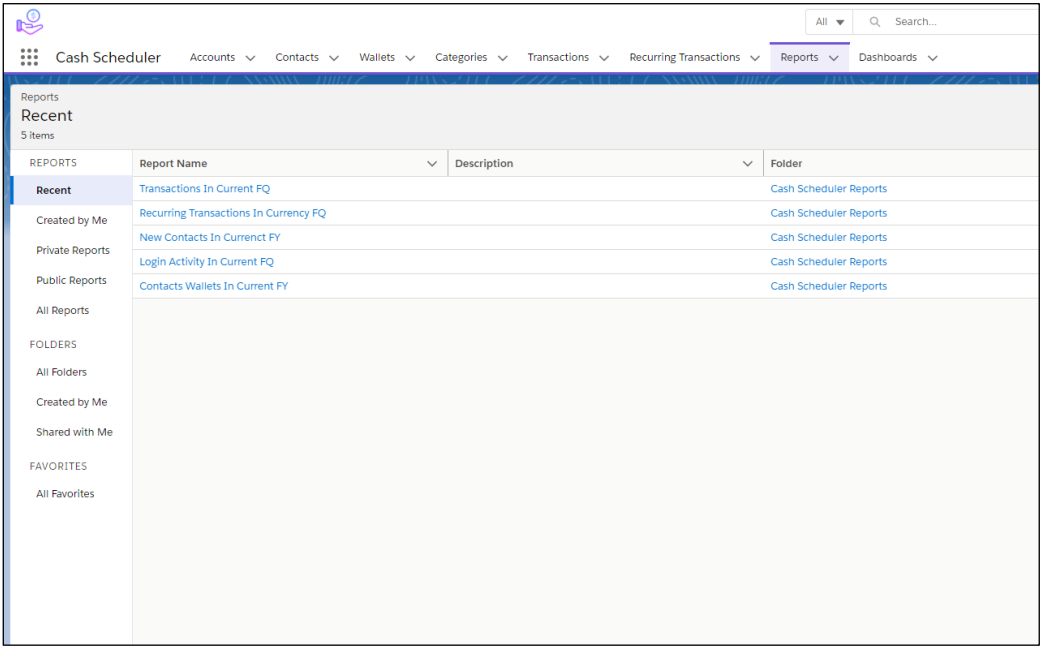


Рисунок 3.3 – Страница с репортами

Репорты строятся исходя из тех данных, которые есть в базе данных *Salesforce* вашей организации. На них можно добавлять все поля, группировать, сортировать и добавлять связанные объекты. Пример репорта представлен на рисунке 3.4.

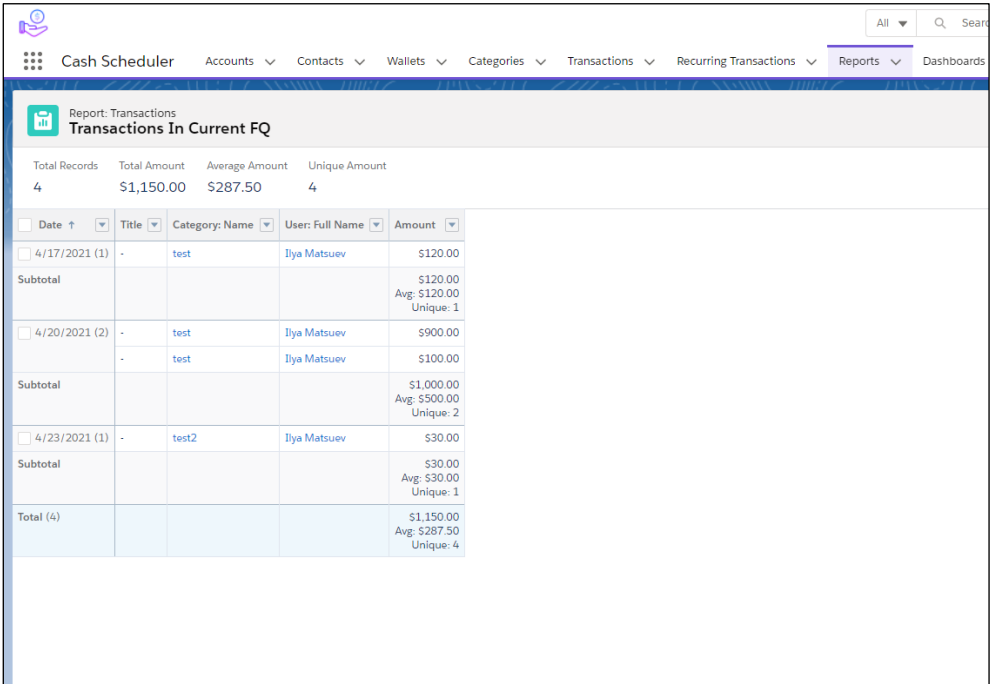


Рисунок 3.4 – Репорт на транзакции в текущем квартале

Репорты можно экспортировать в любом формате, изменять фильтры и сортировать. Однако помимо репортов, хочется видеть визуальное представление информации, для таких вещей есть еще одна вкладка, на которой я создал несколько подборок репортов с графиками. Пример одной из таких подборок представлен на рисунке 3.5.

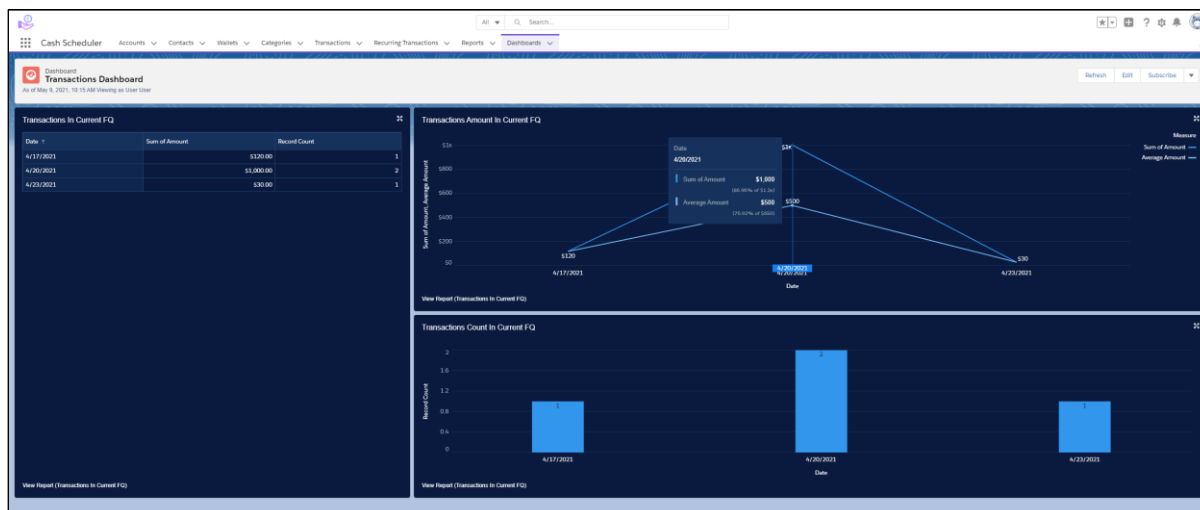


Рисунок 3.5 – Подборка репортов и графиков на транзакции

На данной подборке слева есть тот самый отчет, а справа имеется два графика, один показывает количество транзакций, которые совершили пользователи за определенный день, второй показывает сумму и среднее значение всех транзакций, которые совершали пользователи на конкретную дату.

Подобного рода информация будет полезна для составления определенного рода социологических исследований, что даст возможность искать пути развития продукта. При регистрации пользователей встречается обязательная галочка, говорящая о том, что данные пользователей могут быть использованы для социологических исследований.

Настройка приложения для центра поддержки началось так же с конфигурации внешнего вида и вкладок. На рисунке 3.6 представлена финальная страница центра поддержки.

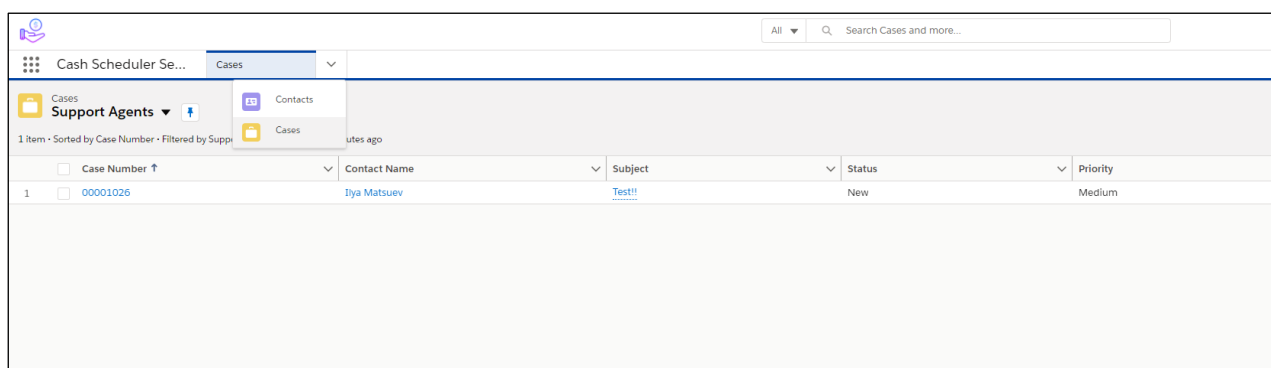


Рисунок 3.6 – Приложение центра поддержки

Как мы видим, для операторов центра поддержки пользователей не предусмотрено вкладок с просмотром информации о репортах и других данных, кроме как пользователей, из того, соображения, что с пользователями им придется общаться и они должны знать их имя и контактную информацию.

Когда агенту поддержки приходит новый кейс, он находится в статусе *New* и остается в очереди кейсов под названием *Support Agents*. Все кейсы, которые пока не были приняты в работу находятся в этой очереди. Когда агент хочет принять кейс в работу, он заходит на него и нажимает *Accept* сверху справа, как показано на рисунке 3.7.

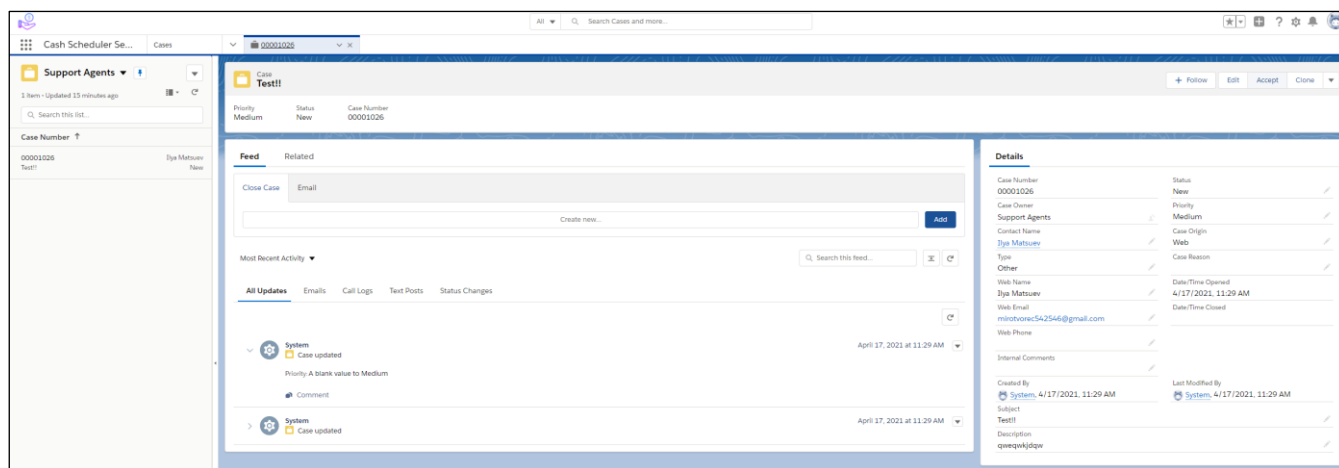


Рисунок 3.7 – Страница кейса

Таким образом этот кейс принимается в работу агентом, устанавливает ему статус *In Progress* и этот кейс убирается из очереди (рисунок 3.8).

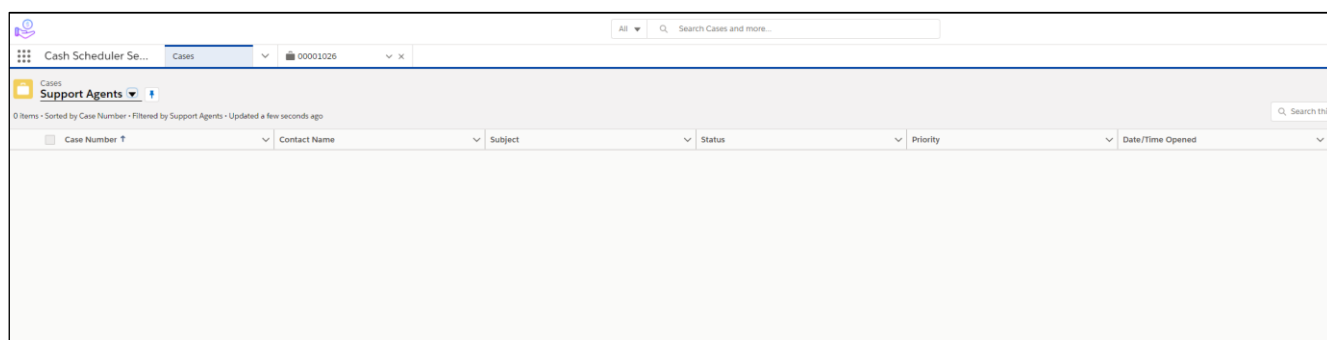


Рисунок 3.8 – Очередь кейсов

Работая с кейсами агент может менять их статус, тип и другие поля, писать под ним комментарии. Когда агент закончил свою работу и готов дать ответ пользователю, это происходит через отправку писем по электронной почте. У агента появляется форма отправки письма с шаблоном, где он может написать свой ответ (рисунок 3.9).

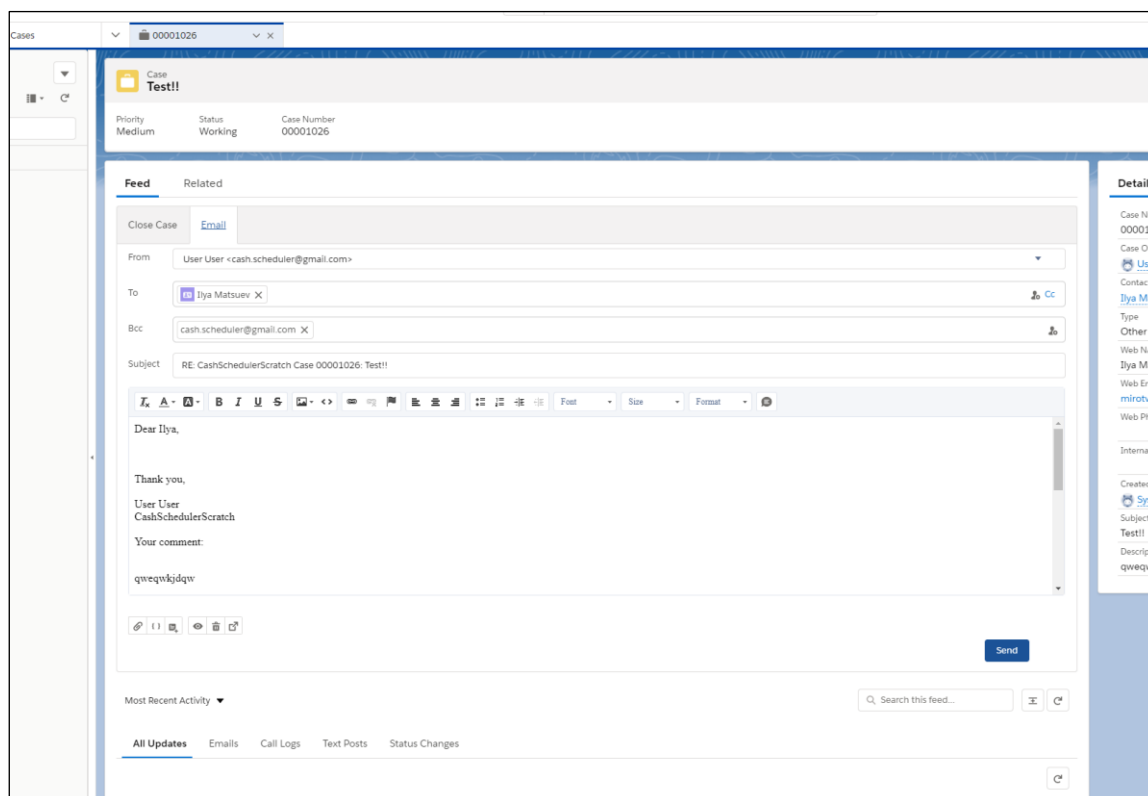


Рисунок 3.9 – Ответ пользователю по почте

После того, как агент отправляет сообщение, оно отправляется не только на почту, но и может как уведомление показываться пользователю прямо в приложении. Для реализации такого функционала был создан триггер на объект *EmailMessage*, который создается после отправки сообщения. После этого отправляется *GraphQL* запрос на *ASP.NET Core* сервер для создания уведомления для пользователя, если пользователь с такой почтой существует. Реализация отправки запроса представлена в листинге 3.24.

```
public with sharing class NotifyCashSchedulerTriggerHandler implements ITriggerHandler {

    public void handle(
        TriggerContext context, ITriggerService service
    ) {
        ICashSchedulerGraphQLService graphQLService
            = new CashSchedulerGraphQLService();
        for (
            Notification notification :
            getNotifications(context.newList)
        ) {
            graphQLService.createNotification(notification);
        }
    }
}
```

```

private List<Notification> getNotifications(
    List<EmailMessage> emailMessages
) {
    List<Notification> notifications = new List<Notification>();
    Map<Id, Case> relatedCases = getRelatedCases(emailMessages);
    for (EmailMessage emailMessage : emailMessages) {
        if (emailMessage.ParentId != null
            &&
            relatedCases.containsKey(emailMessage.ParentId)
            &&
            relatedCases.get(emailMessage.ParentId).ContactId != null
        ) {
            notifications.add(new Notification(
                emailMessage.Subject,
                content,
                relatedCas-
es.get(emailMessage.ParentId).Contact.CashSchedulerId__c
            ));
        }
    }
    return notifications;
}

```

Листинг 3.24 – Реализация отправки запроса на создание уведомления

Здесь создается экземпляр *GraphQL* сервиса для отправки запроса и для каждого письма, которые было отправлено, отправляется свой запрос на создание уведомления.

Таким образом, удалось создать рабочую среду для работы агентов центра поддержки и комфортному общению с клиентом по электронной почте.

3.4 Вывод по разделу

В данном разделе были описаны все ключевые моменты в дипломном проекте, которые необходимы для достижения целей, которые были перечислены в постановке задач и связаны с разработкой приложения.

Созданные резолверы подробно разобраны и описаны в разделе «Описание используемых *GraphQL* резолверов». В приложении В представлена диаграмма классов для одного из резолверов.

Настройка *GraphQL* сервера была подробно описана в разделе «Настройка и установка библиотеки *Hot Chocolate*».

Разработанная интеграция с *Salesforce* подробно описана и показана поэтапно скриншотами в разделе «Настройка *Salesforce* организации».

Итогом выполнения разработки дипломного проекта стало веб-приложение для управления личными финансами. Скриншоты работы готового приложения представлены в приложении Г.

4 Руководство пользователя

4.1 Страница входа

При первом попадании на сайт пользователю будет доступна страница с формами для регистрации, входа и восстановления пароля, а также форма для отправки отзывов или любой обратной связи. Начальное состояние страницы представлено на рисунке 4.1.

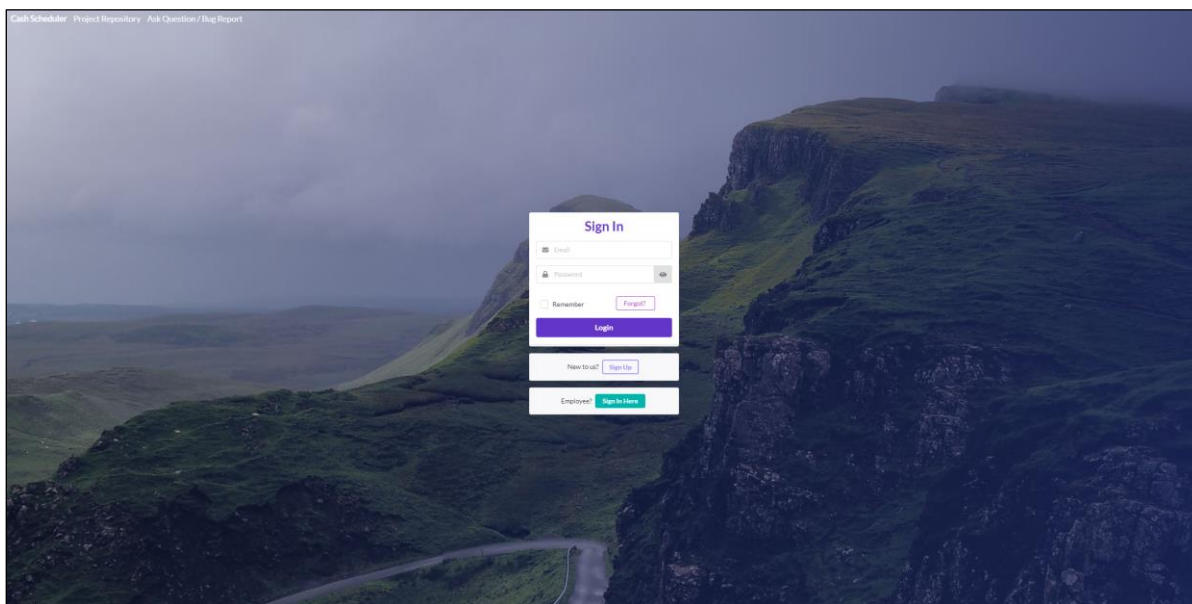


Рисунок 4.1 – Стартовая страница входа

На стартовой странице в первую очередь отображается форма входа. Если пользователь еще не имеет зарегистрированного аккаунта, он может создать его, перейдя на форму регистрации, нажав на кнопку «*Sign Up*». Здесь пользователь может ввести имя, фамилию, баланс, почту и подтвердить пароль. Почта и пароль являются обязательными полями, а тот баланс, который пользователь предоставит, будет использован как баланс кошелька по умолчанию для этого пользователя. При этом для регистрации пользователь должен поставить галочку рядом с пунктом, который говорит, что данные пользователей могут быть использованы для статистики в качестве социального анализа. Эта информация впоследствии будет собираться в *Salesforce* организации и понадобится для анализа аудитории. Форма регистрации представлена на рисунке 4.2.

					БГТУ 04.00.ПЗ		
Изм.	Лист	№ докум.	Подп.	Дата			
Разраб.	Мацуев И.М.				4 Руководство пользователя		
Пров.	Северинчик Н.А.						
Консульт.	Северинчик Н.А.						
Н. контр.	Рыжанкова А.С.						
Утв.	Пацей Н.В.				74417006, 2021		
					Лит.	Лист	Листов
						1	22

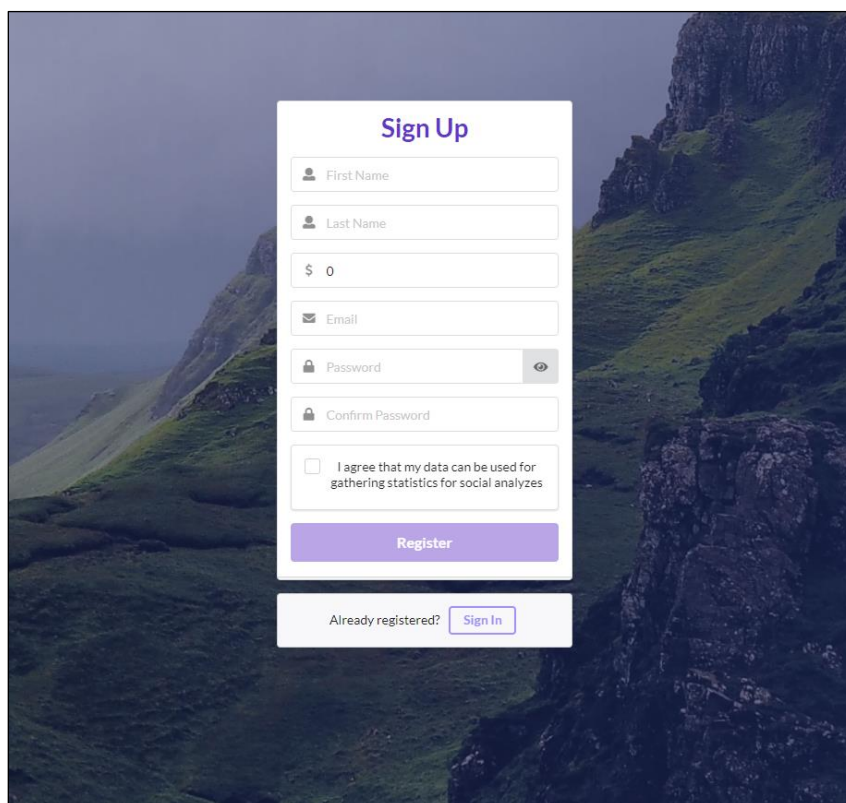
The image shows a 'Sign Up' registration form centered on a background of a mountain landscape. The form is white with a purple title 'Sign Up'. It contains several input fields: 'First Name', 'Last Name', a phone number field with a '\$' icon and the value '0', 'Email', 'Password' (with a toggle eye icon), and 'Confirm Password'. Below these is a checkbox with the text 'I agree that my data can be used for gathering statistics for social analyzes'. A prominent purple 'Register' button is at the bottom of the form. Below the form, there is a link 'Already registered?' followed by a 'Sign In' button.

Рисунок 4.2 – Форма регистрации

В случае, когда пользователь уже имеет аккаунт, но по каким-то причинам не знает пароль от него, он может его поменять через алгоритм подтверждения по электронной почте. Для этого с формы входа необходимо нажать на кнопку «*Forgot?*». Она направит пользователя на форму, где ему будет предложено ввести адрес почты, на который был зарегистрирован аккаунт (рисунок 4.3).

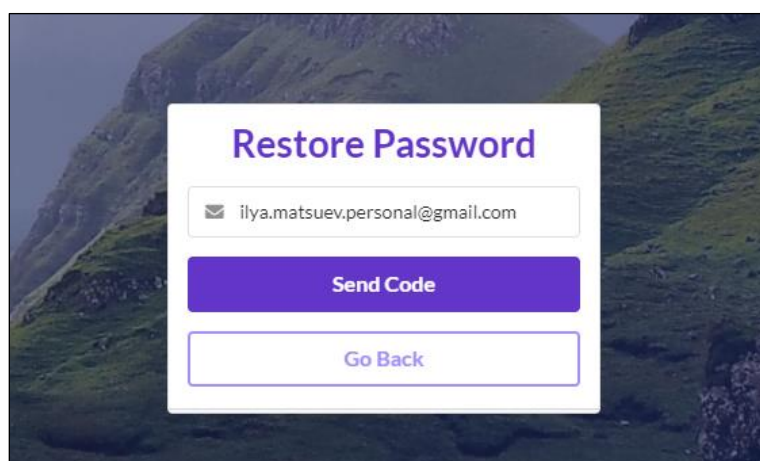
The image shows a 'Restore Password' form centered on the same mountain landscape background. The form is white with a purple title 'Restore Password'. It features an email input field containing 'ilya.matsuev.personal@gmail.com'. Below the field is a large purple 'Send Code' button. At the bottom of the form is a 'Go Back' button with a purple border.

Рисунок 4.3 – Форма ввода почты для восстановления доступа к аккаунту

После этого пользователю необходимо нажать кнопку «*Send Code*», после чего ниже появится еще одно поле для ввода кода подтверждения, который был в тот же момент отправлен на указанную почту. Внешний вид формы представлен на рисунке 4.4.

Рисунок 4.4 – Форма ввода почты для восстановления доступа к аккаунту

После отправки кода, на указанную почту придет сообщение подобно тому, которое представлено на рисунке 4.5.

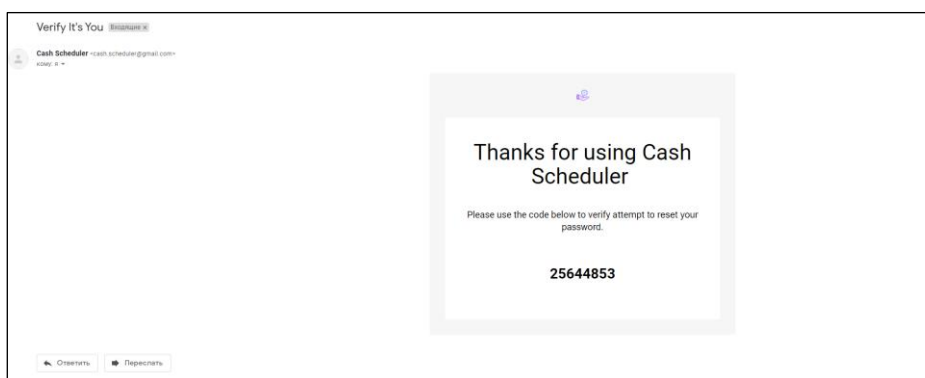


Рисунок 4.5 – Письмо с кодом доступа для восстановления доступа к аккаунту

Данный код может быть использован только в течении трех минут. Плюс к этому, пользователь может запросить новый код только раз в 3 минуты.

Скопировав код из письма пользователь может вернуться на форму и ввести его, если код валидный, то пользователь будет направлен на форму для смены пароля (рисунок 4.6).

Рисунок 4.6 – Форма смены пароля

После того, как он поменяет пароль, он должен войти в аккаунт через форму входа уже с новым паролем.

4.2 Форма отправки обратной связи

Вместе с тем, на стартовой странице присутствует возможность оставлять отзывы или любую обратную связь пользователям. Внешний вид формы представлен на рисунке 4.7.

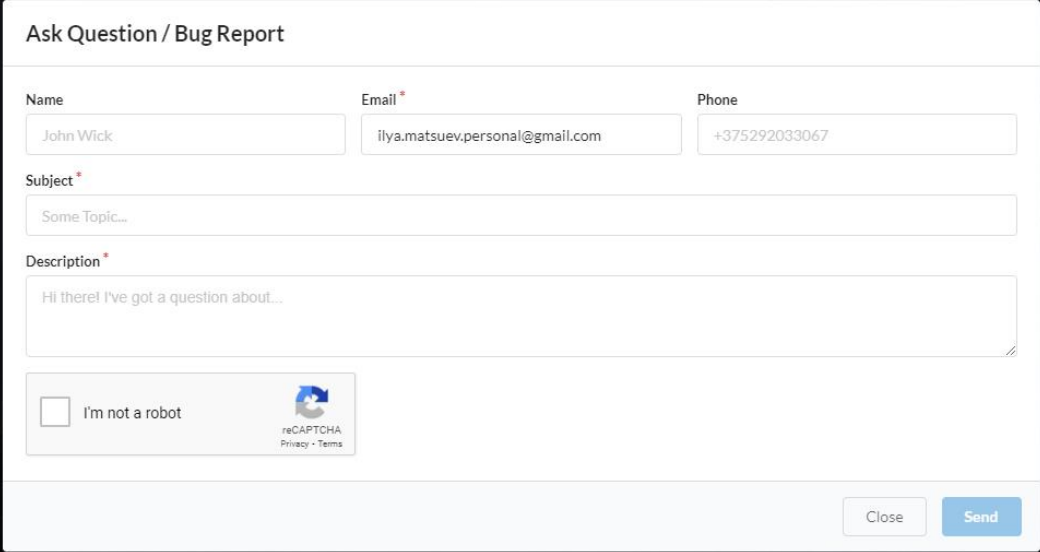


Рисунок 4.7 – Форма обратной связи

На данной форме пользователь должен оставить свои контактные данные. Если пользователь уже вошел в свой аккаунт, то его электронная почта будет предзаполнена. Также здесь необходимо оставить заголовок отзыва или проблемы и их детальное описание. Форма валидируется капчей для того, чтобы избежать спама. После отправки какой-либо обратной связи, пользователю придет письмо извещающее его о том, что его кейс принят на рассмотрение.

4.3 Главная страница

После того, как пользователь вошел в свой аккаунт в первую очередь он попадает на главную страницу приложения, представленную на рисунке 4.8.

Главная страница выполнена в стиле календаря, где пользователь может перемещаться помесечно с помощью кнопок сверху рядом с текущим месяцем и годом. На календаре разным цветом блоков помечены дни предыдущего, текущего и следующего месяцев и сегодняшний день.

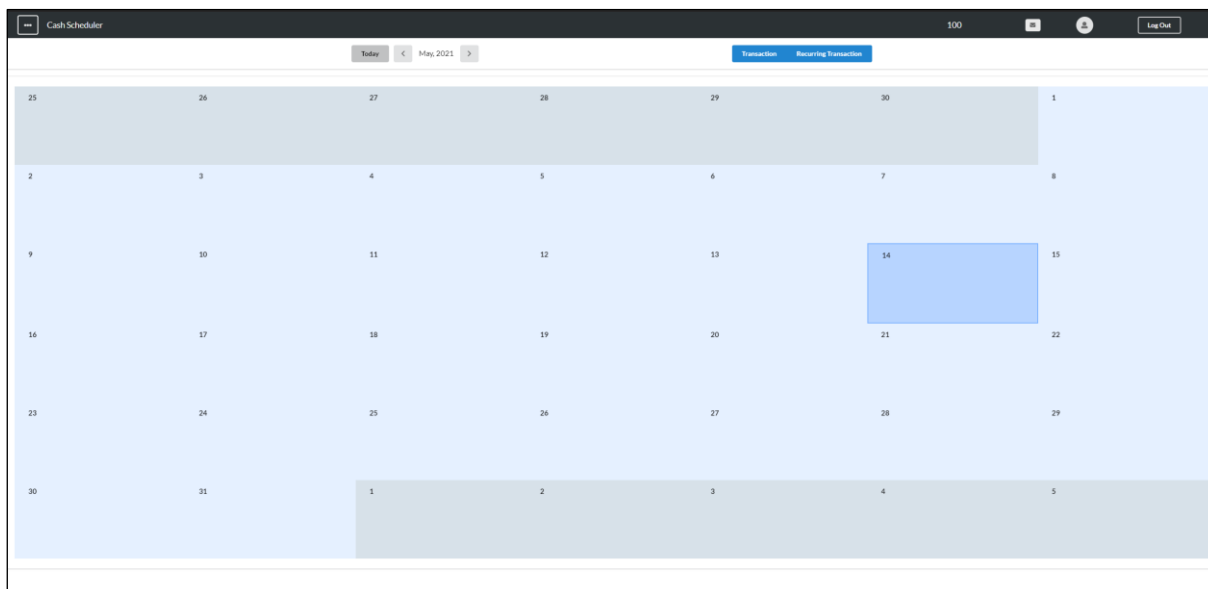


Рисунок 4.8 – Главная страница приложения

По ходу работы пользователя с приложением, на календаре будут добавляться пометки, определяющие сделанные пользователем транзакции. Так, например, если пользователь создаст транзакцию с расходом на 50 долларов, это будет выглядеть на календаре подобно тому, как это показано на рисунке 4.9.

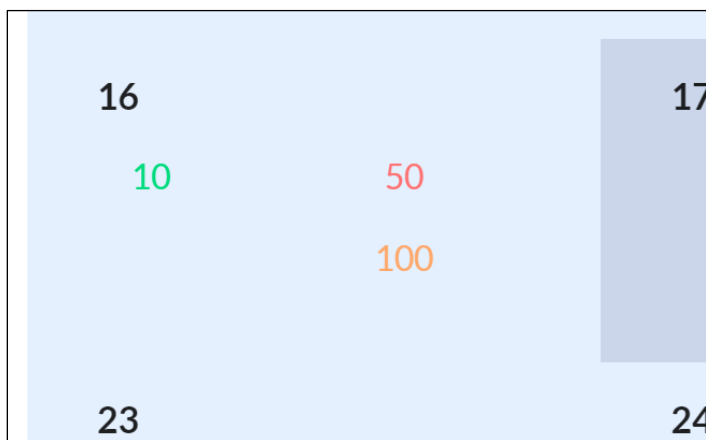


Рисунок 4.9 – Отметка на календаре, показывающая транзакцию расхода

Подобным образом будут отражены суммы транзакций на каждый день. Транзакции дохода отображаются на календаре зеленым цветом, расхода – красным. Запланированные транзакции также отображаются на календаре в соответствии с тем, расходная транзакция или доходная: синим и оранжевым цветами.

Также на главной странице эти самые транзакции можно и создавать с помощью двух кнопок над календарем «*Transaction*» и «*Recurring Transaction*». Первая отвечает за создание одиночной транзакции (например, поход в кино или в магазин). Вторая отвечает за добавление регулярных транзакций, которые будут повторяться в определенном интервале. На рисунке 4.10 показана форма добавления одиночной транзакции.

The screenshot shows a 'New Transaction' form with the following fields: 'Title' (text input), 'Amount' (input with '0'), 'Date' (calendar icon and '2021-05-15'), 'Category' (dropdown menu), and 'Expense' (a button or toggle). Below these is a 'Wallet' dropdown menu. At the bottom right are 'Cancel' and 'Save' buttons. The form is set against a dark background with a light gray border.

Рисунок 4.10 – Форма добавления одиночной транзакции

При добавлении одиночной транзакции обязательными полями являются сумма транзакции, дата, категория и кошелек, к которому эта транзакция привязана. Заголовок может быть не указан.

При выборе категории, нужно также указать тип категории – доход или расход. Подробнее это показано на рисунке 4.11.

This screenshot shows a close-up of the category selection. A dropdown menu is open, showing 'Nitrous Oxide' as the selected category. To its right is a button labeled 'Expense'. Below this, a secondary dropdown menu is visible, showing 'Expense' and 'Income' as options.

Рисунок 4.11 – Выбор типа категории

Каждая транзакция должна быть привязана к какому-либо кошельку, список своих кошельков вы можете увидеть, кликнув на это поле на форме (рисунок 4.12).

This screenshot shows the wallet selection dropdown. The selected wallet is 'USD - Default Wallet'. The dropdown menu is open, showing 'USD - Default Wallet' as the only option.

Рисунок 4.12 – Выбор кошелька

Если вы ничего здесь не укажете, система автоматически выберет ваш кошелек по умолчанию и изменит его баланс в соответствии с суммой транзакции.

Что касается формы для создания регулярных транзакций – там есть все те же поля, что и для одиночной за тем исключением, что дата здесь обозначает

следующую дату, когда эта транзакция должна произойти, а также имеет дополнительное поле, отвечающее за интервал транзакции. Пользователь может настроить транзакцию, чтобы она выполнялась каждый день, еженедельно, ежемесячно или раз в год. Форма создания регулярной транзакции представлена на рисунке 4.13.

Рисунок 4.13 – Форма создания регулярной транзакции

Кроме всего прочего, сверху находится темная шапка, которая имеет в себе элементы, отображающиеся все время, что пользователь находится в приложении, вне зависимости от страницы, на которой он работает. Из элементов здесь можно заметить кнопку выхода (рисунок 4.14), нажав на которую пользователь может выйти из аккаунта. Если он захочет снова войти, то придется опять вводить почту и пароль.



Рисунок 4.14 – Кнопка выхода из аккаунта

Также здесь есть небольшое окошко для просмотра личной информации пользователя (рисунок 4.15). Здесь он может сменить свое имя, фамилию и баланс кошелька по умолчанию. Электронная почта не может быть изменена, так как она является еще и логином для входа в аккаунт. После того, как пользователь ввел все необходимые изменения, ему остается нажать на кнопку «Save».

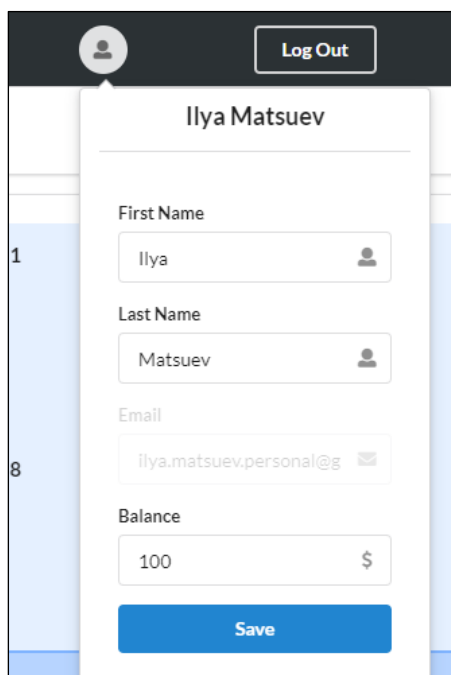


Рисунок 4.15 – Окно просмотра информации пользователя

Кроме просмотра информации о пользователе здесь также имеется окошко для просмотра уведомлений в приложении, показанное на рисунке 4.16.

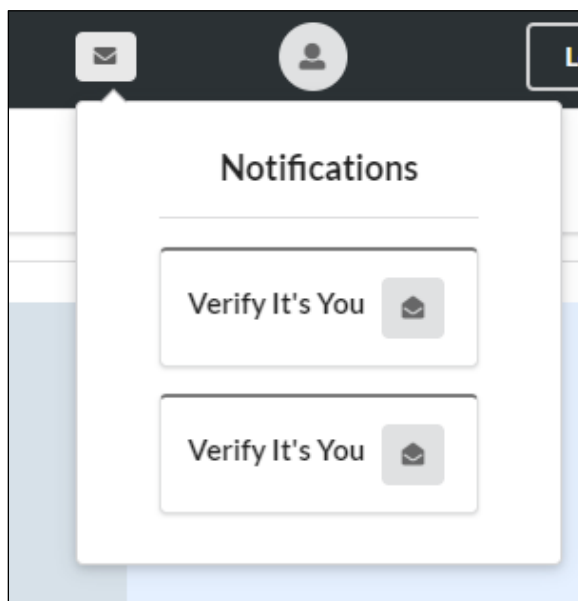


Рисунок 4.16 – Окно просмотра уведомлений пользователя

Например, вместо того, чтобы заходить на свою почту, для подтверждения смены пароля, пользователь может посмотреть это же сообщения у себя в приложении, если он вошел, например, с другого браузера или устройства. Для того, чтобы пометить уведомление как прочитанное или не прочитанное, нужно нажать на кнопку с изображением конверта прямо на уведомлении. Пример раскрытого уведомления показан на рисунке 4.17.

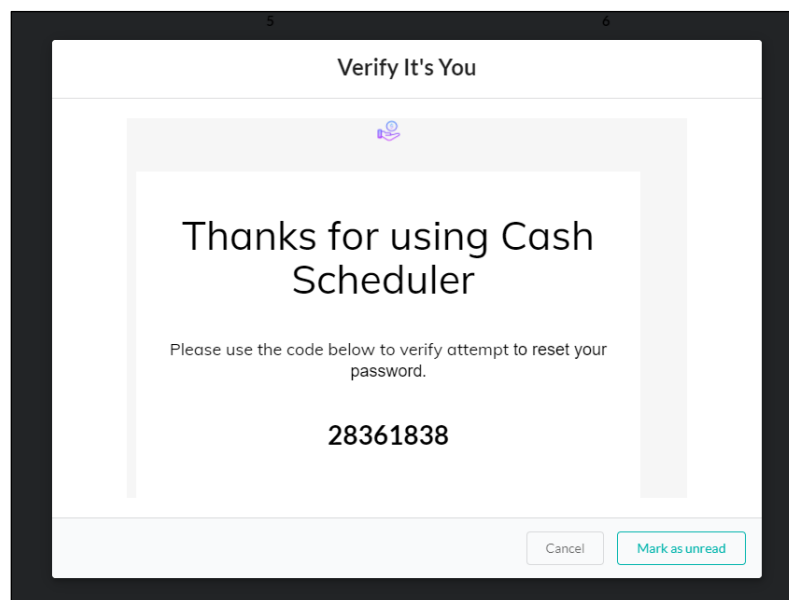


Рисунок 4.17 – Открытое уведомление пользователя

Уведомление отображается точно также, как и то, что пришло на почту.

Также в шапке страницы по умолчанию отображается баланс пользователя, при нажатии на который, его перекинет на страницу просмотра детальной информации о транзакциях. Однако, если пользователю это число сверху каким-то образом мешает, его, как и окно с уведомлениями, можно убрать соответствующей конфигурацией в настройках приложения.

Последним составляющим шапки приложения является меню выбора страницы, представленное в раскрытом состоянии на рисунке 4.18.

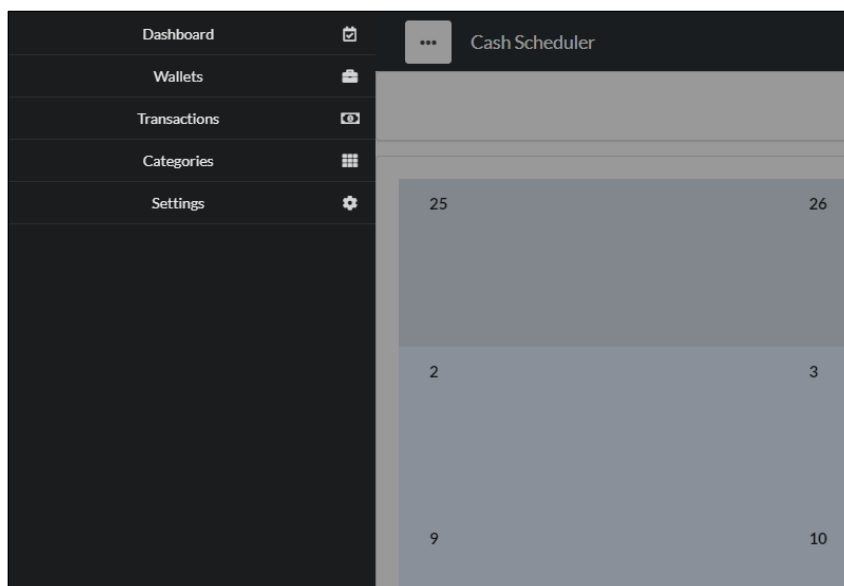


Рисунок 4.18 – Открытое меню приложения

Здесь можно перейти на любую из страниц, существующих в приложении. В данный момент мы находимся на главной и самой первой странице в списке. Следующая страница это страница со всеми кошельками пользователя.

4.4 Страница просмотра и редактирования кошельков

Используя меню приложения мы можем перейти на вторую страницу в списке – страницу просмотра и редактирования кошельков пользователя, показанную на рисунке 4.19.

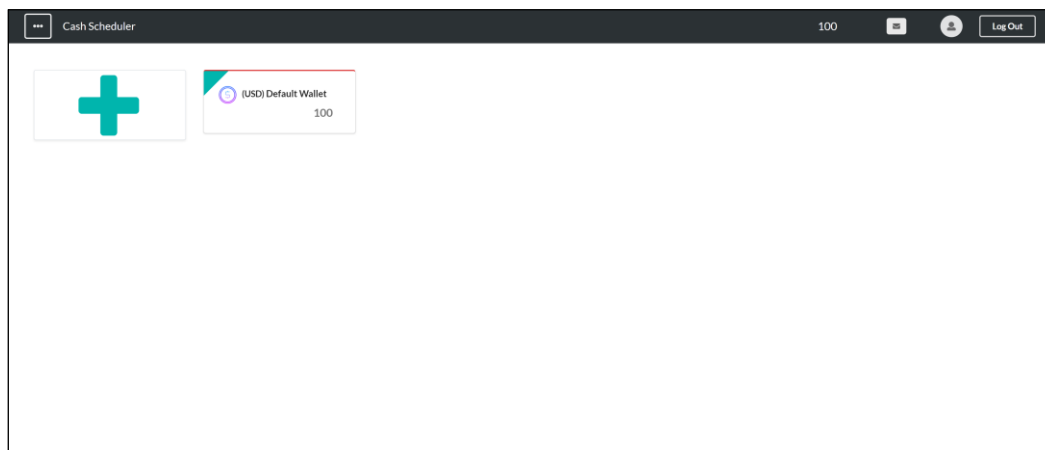


Рисунок 4.19 – Страница с кошельками пользователя

На данной странице пользователь может:

- создавать новые кошельки;
- редактировать существующие кошельки;
- удалять кошельки;
- делать переводы между кошельками.

Для создания нового кошелька пользователю просто нужно нажать на большой плюс. Ему выведет форму создания кошелька, которая представлена на рисунке 4.20.

Рисунок 4.20 – Создание нового кошелька

На форме пользователю необходимо ввести имя кошелька, его баланс и валюту. Также можно пометить его как стандартный кошелек, что будет означать, что при создании транзакций этот кошелек будет выбираться кошельком по

умолчанию. В поле с выбором валюты представлены все существующие валюты на данный момент.

Для редактирования кошелька пользователь может просто щелкнуть по нему мышкой и появится точно такая же форма, как и при создании. Однако, если мы захотим поменять валюту уже существующего кошелька, то добавятся новые поля, показанные на рисунке 4.21.

Рисунок 4.21 – Редактирование кошелька с изменением валюты

Здесь пользователь может выбрать – нужно ли ему конвертировать уже существующий баланс на кошельке в другую валюту. Если нет, достаточно просто перевести галочку «*Convert Balance?*» в неактивное состояние и сохранить. Если это необходимо, то пользователь должен выбрать курс обмена валюты, который будет использован для конвертации.

Рисунок 4.22 – Выбор курса обмена валют

Когда пользователь нажимает на это поле, самый первый вариант, который ему предлагается – это курс европейского банка, полученный через *API* с сервера. Если пользователь хочет сделать это с каким то другим курсом, то он может

добавить свой собственный, просто вписав нужный коэффициент в поле и нажав на кнопку «*Add Exchange Rate: ...*». После этого пользователь сможет использовать этот курс обмена при совершении других валютных операций в приложении с такими валютами. Визуально это показано на рисунке 4.23.

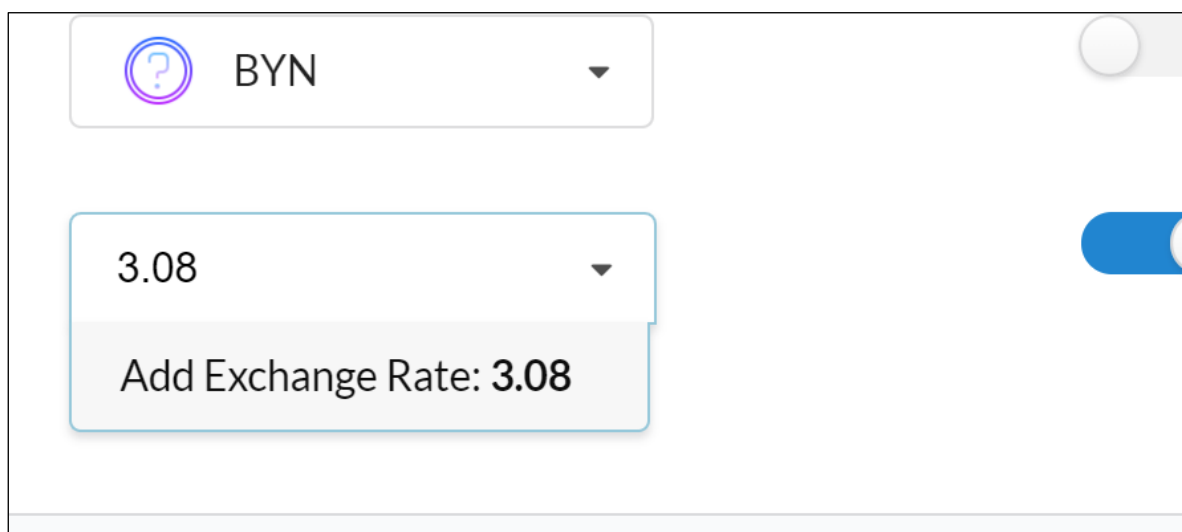


Рисунок 4.23 – Добавление собственного курса обмена валют

После этого, при сохранении, баланс кошелька будет обновлен в соответствии с исходными средставми и курсом обмена.

Для удаления кошелька нужно просто нажать на кнопку «*Delete*» в той же форме редактирования. После этого у пользователя будет запрошено подтверждение, так как при удалении кошелька, будут удалены и все транзакции, которые с ним были связаны (рисунок 4.24.).

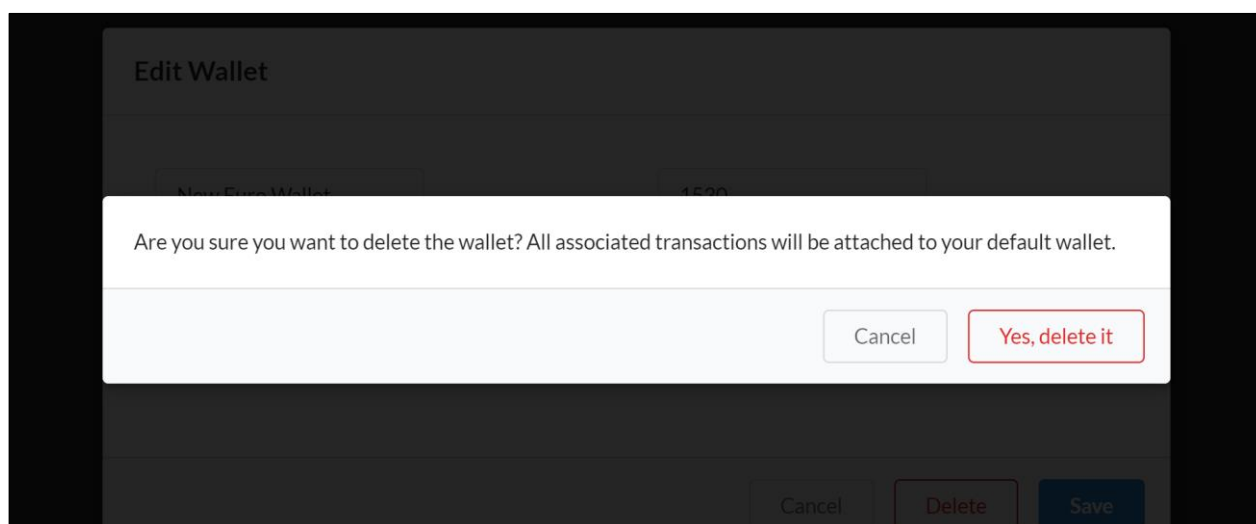


Рисунок 4.24 – Подтверждение удаления кошелька

Если пользователя это устраивает, он может нажать на кнопку «*Yes, delete it*». После этого все его транзакции будут удалены вместе с этим кошельком.

Для выполнения трансфера между двумя кошельками, достаточно просто навести один кошелек на другой, как показано на рисунке 4.25.

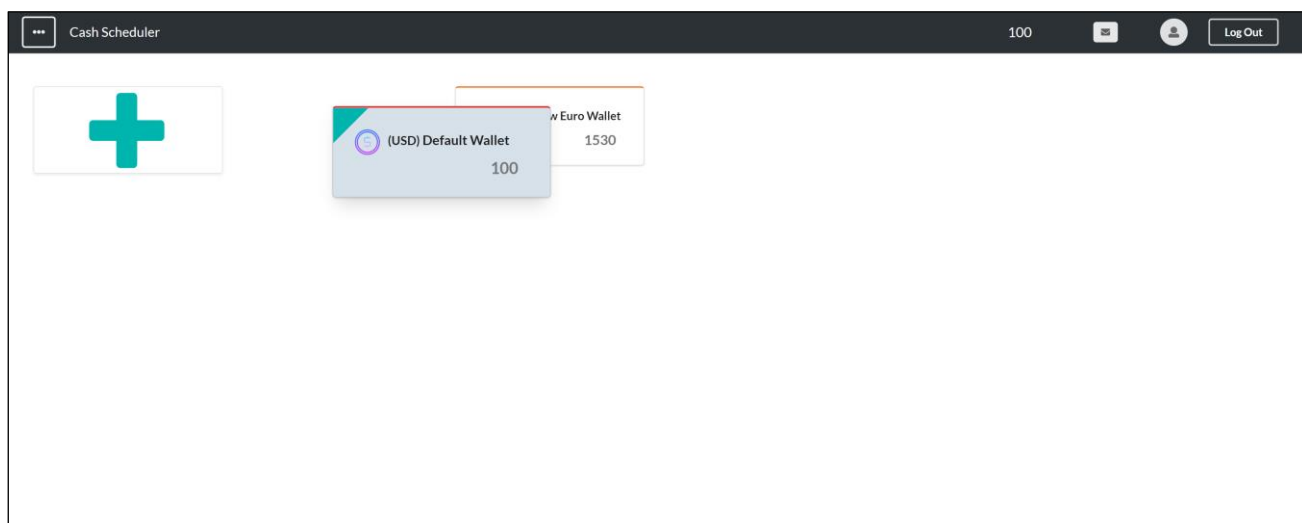


Рисунок 4.25 – Перевод с одного кошелька на другой

После этого действия пользователю будет представлена форма для оформления перевода с кошелька, который был перетянут на кошелек, над которым курсор был опущен (рисунок 4.26). Однако на этой форме также можно и выбрать нужные кошельки вручную.

The screenshot displays a "New Transfer" form. At the top, it has the title "New Transfer". Below the title, there are two dropdown menus for selecting wallets: the first is "USD - Default Wallet" with a dollar sign icon, and the second is "BYN - New Euro Wallet" with a question mark icon. Under these, there is a text input field containing the number "0". Below the input field is another dropdown menu labeled "Exchange Rate". At the bottom right of the form, there are two buttons: a light gray "Cancel" button and a blue "Transfer" button.

Рисунок 4.26 – Форма перевода средств

Кроме того, здесь необходимо ввести сумму перевода и курс обмена валют. Курс обмена работает точно также, как и при попытке изменения валюты у

существующего кошелька, что было описано в этом разделе выше. Когда все поля заполнены, пользователь может нажать кнопку «*Transfer*» для совершения перевода.

После совершения перевода, балансы кошельков обновляются в соответствии с суммой и курсом, который был указан в форме.

4.5 Страница просмотра и изменения категорий

При создании транзакций пользователю необходимо указать, к какой категории будет относиться его транзакция. Существуют как стандартные транзакции, которые по умолчанию для выбора, так и пользовательские категории, которые пользователь может создавать сам для своих собственных целей. Для этого была разработана страница для просмотра и изменения категорий. По умолчанию пользователю показывается список стандартных категорий. Начальный вид страницы представлен на рисунке 4.27.

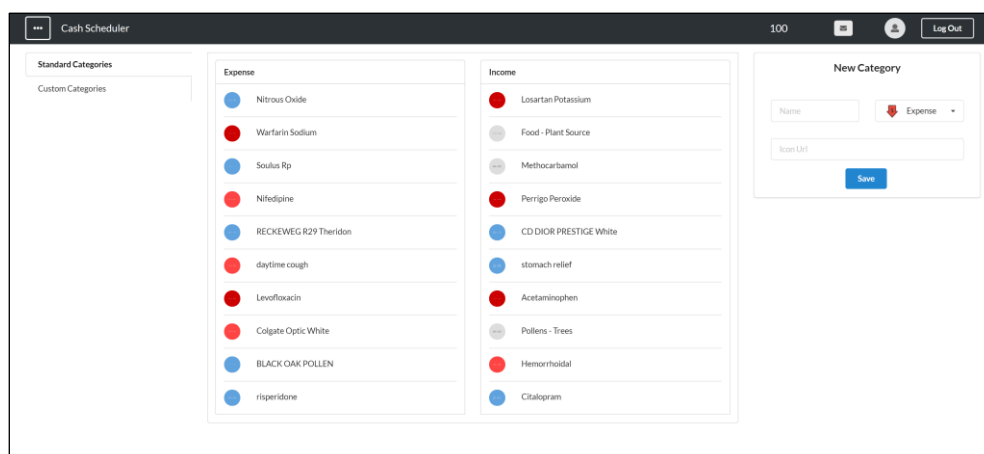


Рисунок 4.27 – Страница просмотра и изменения стандартных категорий

На странице представлен список категорий в две колонки, одна с расходами, другая с доходами. Для просмотра списка созданных пользователем категорий, нужно нажать на раздел справа (рисунок 4.28).

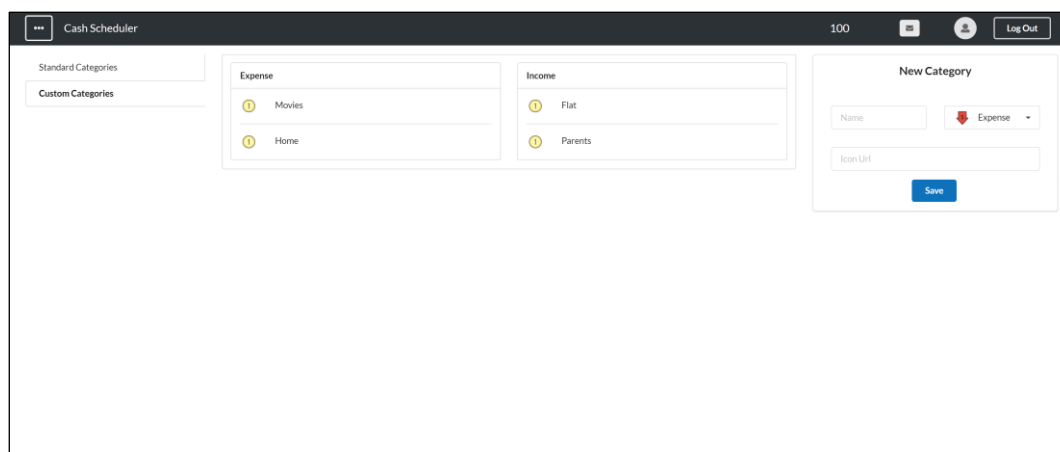


Рисунок 4.28 – Страница просмотра и изменения пользовательских категорий

На этой же странице пользователь может и создавать категории. Делается это с помощью формы, которая расположена справа. Из полей для заполнения здесь только имя категории, ее тип и ссылка на иконку, если пользователь хочет поставить свою собственную иконку для категории. Если ссылка для иконки не задана, она будет предзаполнена иконкой по умолчанию. Детальнее форма представлена на рисунке 4.29.

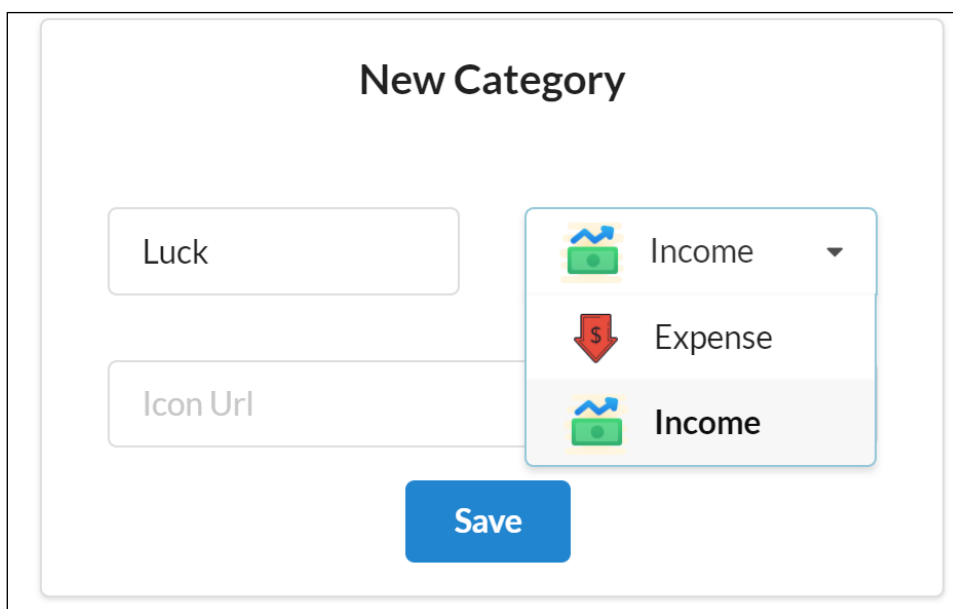


Рисунок 4.29 – Форма создания категории

После создания категории, она будет показана в списке со всеми остальными пользовательскими категориями. Для изменения названия или иконки категории, можно просто кликнуть по нужной категории из списка и появится форма для изменения этих полей (рисунок 4.30).

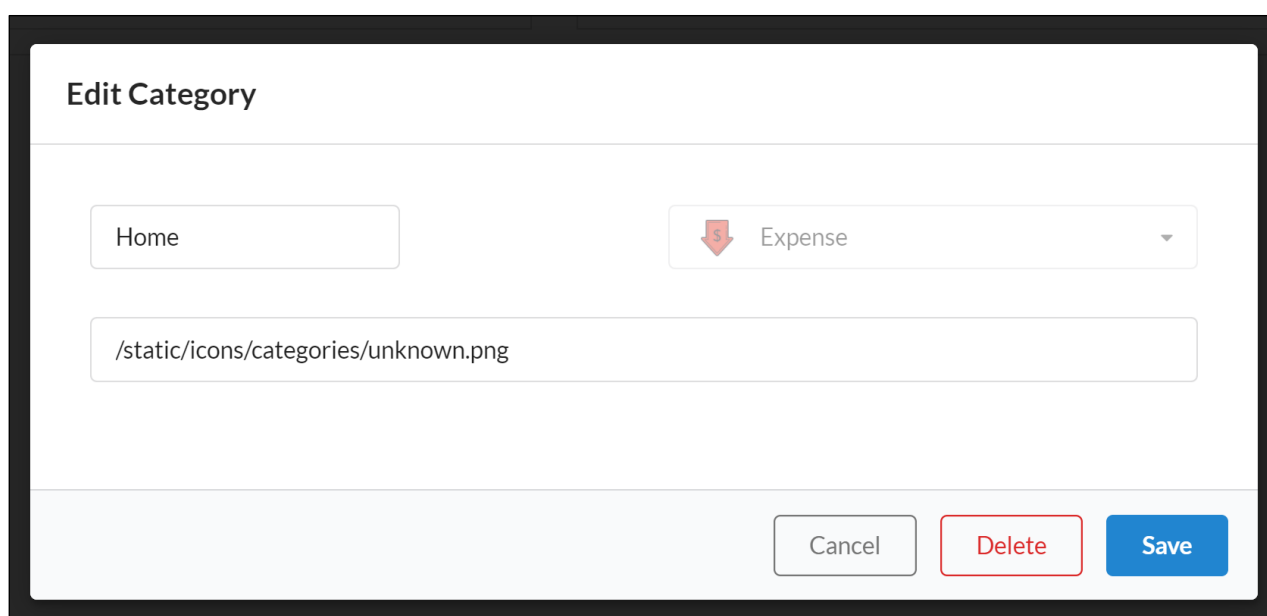


Рисунок 4.30 – Форма изменения категории

Изменение типа категории не допускается, однако пользователь может поменять ее имя и ссылку на иконку. Также на форме есть кнопка для удаления категории. При попытке удаления, пользователь будет уведомлен о том, что при удалении категории, все связанные транзакции также будут удалены.

4.6 Страница с детальным описанием транзакций

Все приложение полностью завязано на функционале вокруг транзакций. Транзакция – это любая денежная операция, будь то расход или доход. В приложении присутствует страница для просмотра списка своих транзакций и небольшой сводки по ним за период времени. Начальный вид страницы изображен на рисунке 4.31.

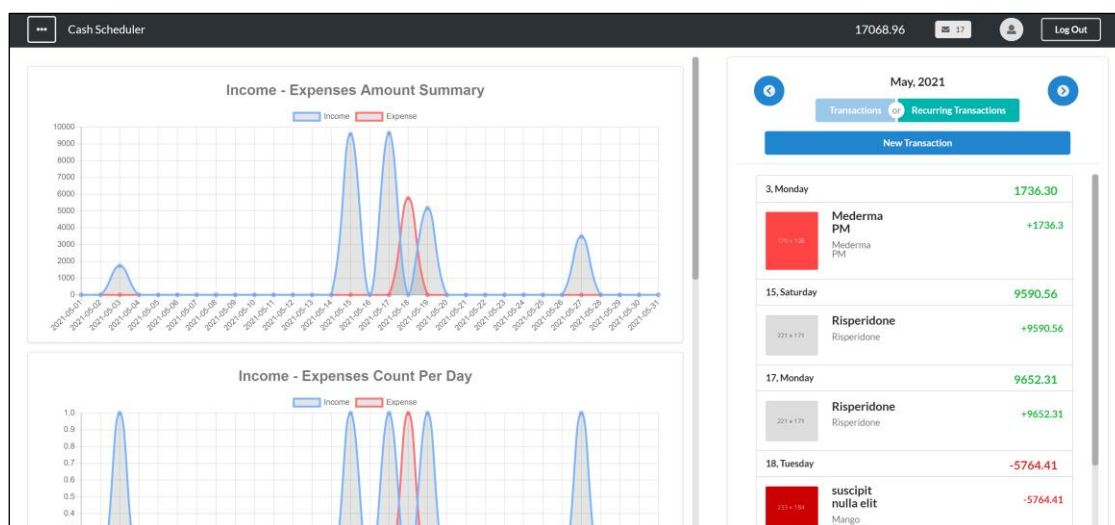


Рисунок 4.31 – Начальная страница просмотра транзакций

На данной странице представлено несколько графиков по центру страницы и список транзакций, сгруппированных по месяцам и дням справа. Рассмотрим сначала графики. Первый из них показывает соотношение размеров доходов и расходов от начала и до конца текущего месяца (рисунок 4.32).

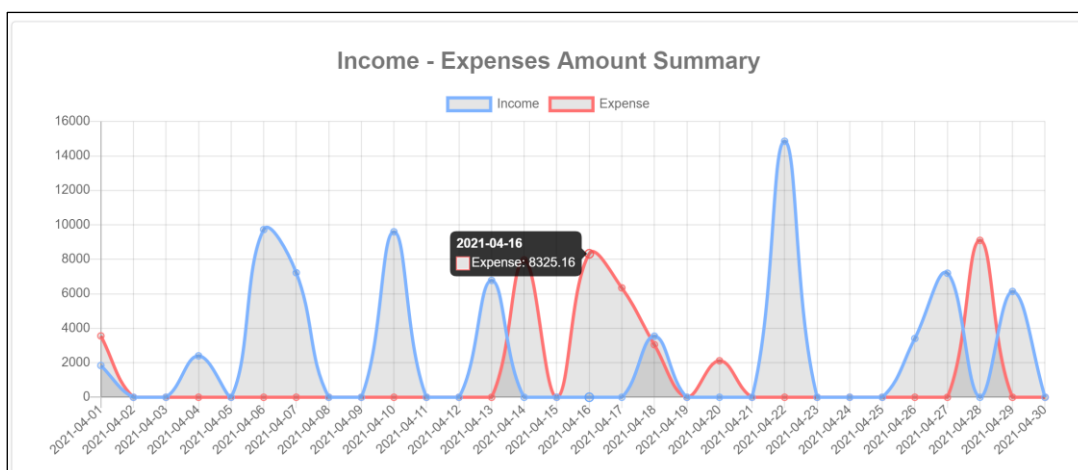


Рисунок 4.32 – График соотношений доходов и расходов за текущий месяц

Следующий график находится прямо под первым и показывает уже количество расходных и доходных транзакций ежедневно за текущий месяц (рисунок 4.33).

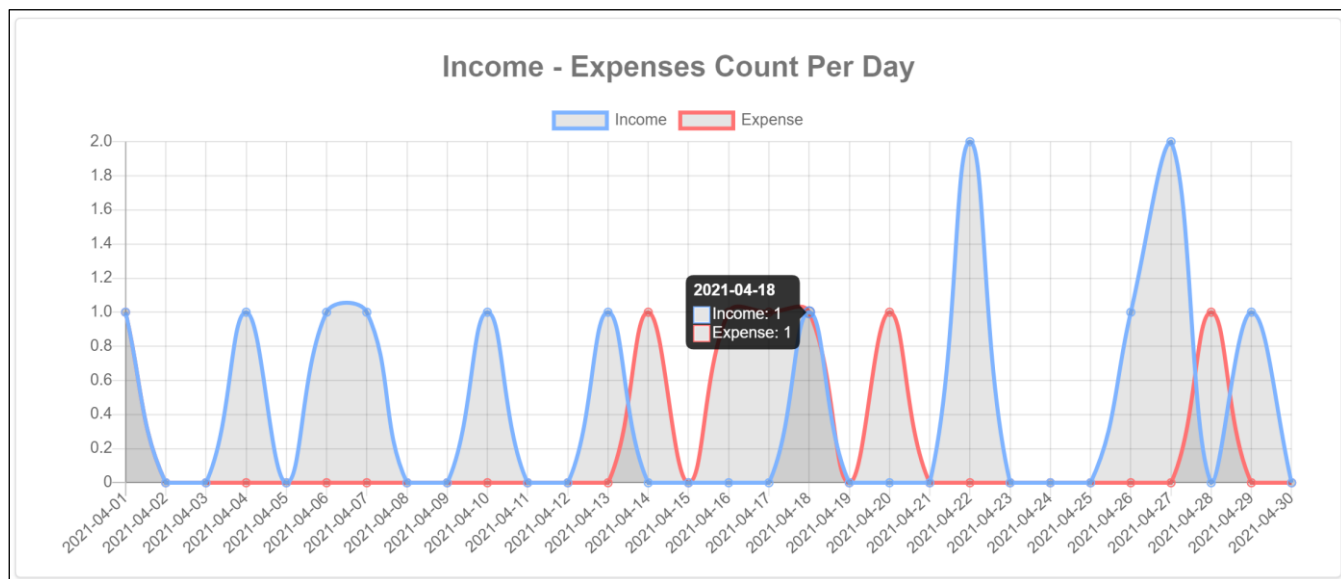


Рисунок 4.33 – График соотношений количества транзакций

Третий разделен на две части. Первая показывает соотношение категорий транзакций дохода друг к другу, для определения более приоритетного и наибольшего источника доходов за текущий месяц. И еще один такой же для категорий расходов (рисунок 4.34).



Рисунок 4.34 – Графики соотношений сумм доходных и расходных категорий

Ну и последний график показывает прогрессию сумм всех транзакций за последний год (рисунок 4.35). Суммы расходов и доходов складываются ежемесячно и изображаются для сравнения на годовом графике. Таким образом можно увидеть, насколько больше или меньше пользователь стал тратить. Например, если пользователь имеет суммарно транзакций дохода больше, чем транзакций расхода, то в этом месяце колонка будет выше нуля. По такой же логике вычисляются все остальные месяцы, и в итоге можно наглядно видеть насколько финансово менялась ситуация за последний год.

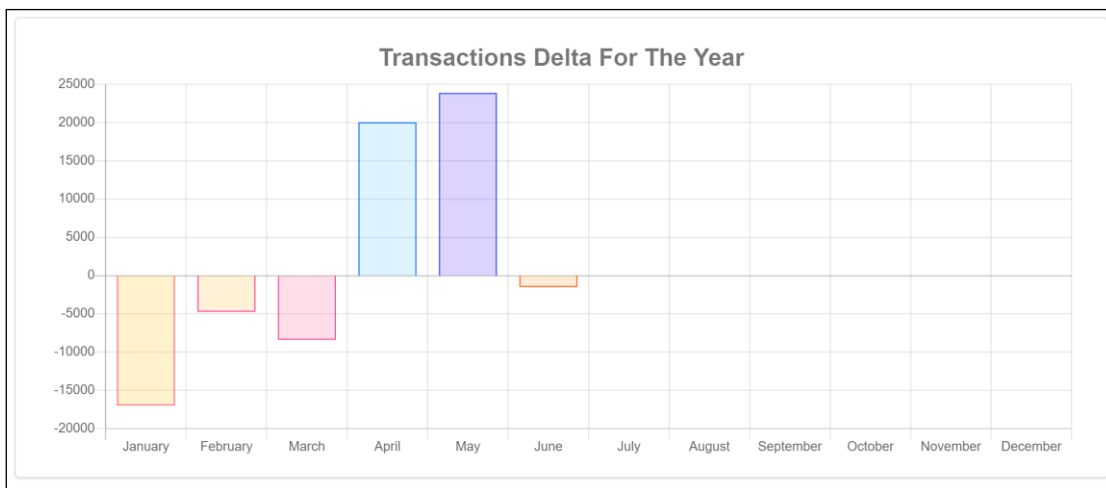


Рисунок 4.35 – График просмотра прогрессии сумм транзакций за год

Теперь перейдем к секции справа. Первое, для чего она нужна, это для просмотра списка транзакций сгруппированного по дням (рисунок 4.36).

May, 2021		
<div> <div>Transactions</div> <div>or</div> <div>Recurring Transactions</div> </div>		
New Transaction		
3, Monday		1736.30
<div>179 x 198</div> <div>Mederma PM</div> <div>Mederma PM</div>		+1736.3
15, Saturday		9590.56
<div>221 x 171</div> <div>Risperidone</div> <div>Risperidone</div>		+9590.56
17, Monday		9652.31
<div>221 x 171</div> <div>Risperidone</div> <div>Risperidone</div>		+9652.31
18, Tuesday		-5764.41
<div>233 x 184</div> <div>suscipit nulla elit</div> <div>Mango Blossom</div>		-5764.41

Рисунок 4.36 – Список транзакций за месяц

Чтобы показать транзакции другого месяца, нужно просто нажать на соответствующую кнопку вправо или влева в шапке секции. При смене месяца, все данные на графиках (если они визуализируют информацию за месяц) будут также обновлены. Также здесь можно переключить показ для просмотра повторяющихся транзакций. Это можно сделать нажав на кнопку «*Recurring Transactions*». В добавок прямо здесь можно и создавать новые транзакции. Форма создания транзакций точно такая же, как и на главной странице приложения.

Нажав на любую из транзакций появится форма редактирования транзакции, показанная на рисунке 4.37. Для редактирования пользователю доступны только такие поля как заголовок транзакции, сумма и дата транзакции. На рисунке видно, что поля, которые недоступны для редактирования заблокированы для изменения пользователем.

Рисунок 4.37 – Форма редактирования транзакции

Здесь также можно и удалить транзакцию. Стоит отметить, что создание транзакций, изменение их суммы или даты, удаление транзакций влияет на баланс кошелька к которому они привязаны. Если пользователь создает транзакцию, с датой в будущем времени, то в данный момент на баланс кошелька это никак не повлияет. Транзакция будет применена в тот момент, когда ее дата наступит. Если же пользователь создает транзакцию в прошлом, то с кошелька будет списана необходимая сумма сразу же. При изменении суммы транзакции или ее удалении сумма кошелька также будет меняться в зависимости от того, находится транзакция в прошлом или она запланирована.

На рисунке 4.38 представлена форма редактирования повторяющихся транзакций.

Рисунок 4.38 – Форма редактирования транзакции

Она практически полностью повторяет функциональность такой же формы для одиночных транзакций за тем исключением, что здесь нельзя менять дату следующего применения транзакции.

4.7 Страница просмотра и изменения настроек пользователя

Кроме всего прочего в приложении предусмотрена возможность изменения некоторых настроек для более удобной работы пользователя с ним. Для этих целей была разработана отдельная страница со всеми доступными настройками пользователя. Начальный вид страницы настроек представлен на рисунке 4.39.

Рисунок 4.39 – Страница настроек пользователя

Настройки имеют три раздела: общие настройки, настройки уведомлений и раздел интеграций. По умолчанию пользователь попадает на страницу с общими настройками. Среди них есть такие настройки как включение/выключение отображения баланса в шапке страницы, выбор языка интерфейса приложения, переключение в режим темной темы и возможность полного удаления аккаунта

пользователя. На рисунке 4.40 показан результат применения всех перечисленных выше настроек, кроме удаления аккаунта пользователя, разумеется.

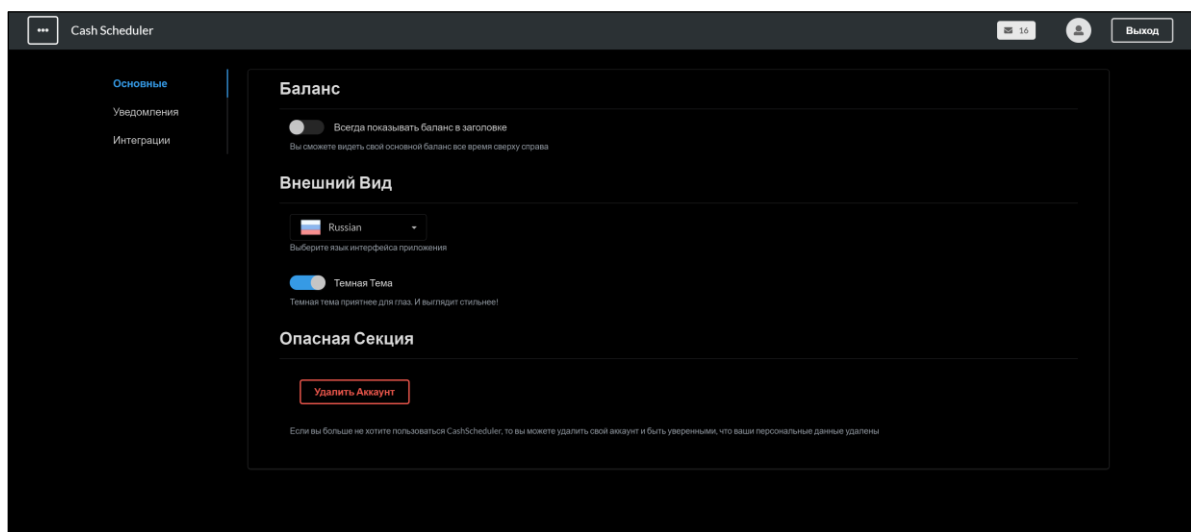


Рисунок 4.40 – Результат применения общих настроек

Как видно на рисунке выше, из шапки приложения пропал баланс основного кошелька, язык интерфейса во всем приложении поменялся на русский и темная тема была активирована.

На рисунке 4.41 показан раздел настроек уведомлений, который содержит в себе настройки каким-либо образом связанные с уведомлениями в приложении. Можно запросить дублировать уведомления на почту, выключить звук приходящих уведомлений, либо вовсе выключить функцию уведомлений, если она пользователю не нужна.

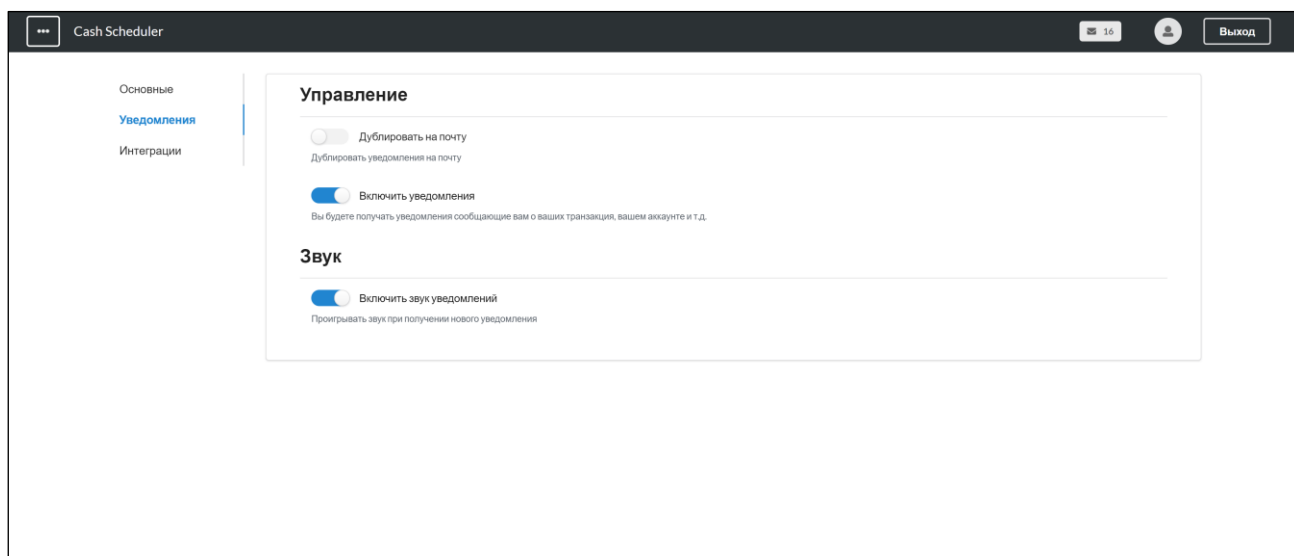


Рисунок 4.41 – Раздел настроек уведомлений

Последний раздел настроек это раздел интеграций. Он представлен на рисунке 4.42.

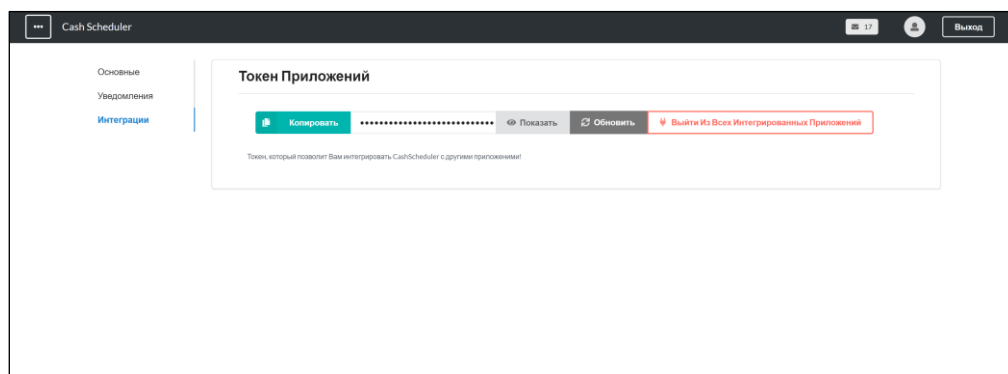


Рисунок 4.42 – Раздел настроек интеграций

В данном разделе настроек присутствует только одна секция, которая предназначена для работы с токеном приложений. Токен приложений – это специальный токен пользователя, который он может предоставлять сторонним приложениям для предоставления им возможности создания категорий, транзакций и кошельков для пользователя автоматически с помощью *GraphQL API*. В данной секции пользователь может скопировать токен нажав соответствующую кнопку, показать его, обновить, если необходимо, либо выйти из всех интегрированных приложений, таким образом сторонние приложения, использующие этот токен не смогут обновлять свои токены и потеряют доступ к аккаунту пользователя.

4.8 Выводы по разделу

В данном разделе представлена инструкция для пользователя по работе с веб-приложением. Она предоставляет описание основных возможностей, которые можно совершить при работе с приложением.

Пользователю доступны регистрация, вход, а также восстановление пароля по электронной почте. При работе с приложением, пользователи могут создавать обновлять и удалять категории, транзакции и кошельки, а также изменять настройки работы приложения для своего аккаунта. Пользователи имеют возможность удобно просматривать созданные или запланированные ими транзакции, видеть статистику по ним и делать из этого выводы. Были разработаны несколько графиков, визуализирующих данные о транзакциях пользователей за определенный промежуток времени. Кроме всего прочего у пользователей есть возможность создавать повторяющиеся транзакции, которые будут сами создаваться по заданному пользователем интервалу. Баланс кошельков меняется автоматически в зависимости от сумм и дат транзакций, что позволяет пользователю не редактировать его актуальный баланс часто. Стоит отметить, что в приложении также была реализована функция оповещений, которая позволяет пользователю не покидать приложения для просмотра уведомлений, касающихся его аккаунта, не заходя на электронную почту.

Если пользователь больше не планирует пользоваться приложением, он может с легкостью удалить свой аккаунт и все данные из соображений безопасности.

5 Тестирование веб-приложения

Тестирование одна из самых важных частей разработки веб-приложения. Именно в процессе тестирования находят подавляющее большинство проблем в работе.

Необходимо протестировать максимальное число вариантов использования веб-приложения пользователем, чтобы исключить возможные ошибки в процессе работы приложения.

Среди использованных подходов к тестированию можно отметить *unit* и *manual* тестирование, результаты которых будут описаны в данном разделе.

5.1 Unit тестирование

Unit тестирование, или как оно еще называется, модульное тестирование, направлено на проверку отдельных модулей исходного кода, которое позволяет проверять не привело ли какое-либо изменение кода к появлению ошибок. В случае появления ошибки, такой вид тестирования позволяет достаточно быстро их обнаружить и устранить, поскольку затрагивает только один модуль.

В данном проекте тестами были покрыты все сервисы серверной части *ASP.NET Core* приложения, а также весь *Apex* код *Salesforce* организации.

Для покрытия тестами *ASP.NET Core* приложения, я использовал библиотеку *XUnit*, которая имеет все инструменты для написания качественных тестов на языке *C#*. *XUnit* является более современным по отношению к его самому главному конкуренту *NUnit*. Обе библиотеки обладают хорошей документацией и широкой базой пользователей, однако *XUnit* сейчас более широко используется для написания тестов под платформу *ASP.NET Core* благодаря многочисленным изменениям в плане написания кода, тестовых проверок и того, как сами тесты запускаются.

Salesforce, в свою очередь, имеет свою политику, касающуюся юнит-тестирования. Перед тем, как можно будет задеплоить код или пакет, необходимо, чтобы весь *Apex* код был покрыт тестами как минимум на 75% с учетом того, что все тесты должны выполняться успешно. Таким образом, *Salesforce* просто не позволяет не писать юнит-тесты, иначе разрабатываемое веб-приложения рискует остаться на той среде, в которой и разрабатывалось.

Всего был разработан 71 юнит тест для *ASP.NET Core* приложения и еще 6 тестов, которые покрывают *Apex* код, разработанный в течении дипломного проекта. Описание основных тестовых методов (по одному из каждого сервиса) и модулей, которые они тестируют, представлено в таблице 5.1.

					БГТУ 05.00.ПЗ		
Изм.	Лист	№ докум.	Подп.	Дата			
Разраб.	Мацуев И.М.				5 Тестирование веб-приложения	Лит.	Лист
Пров.	Северинчик Н.А.						Листов
Консульт.	Северинчик Н.А.						1
Н. контр.	Рыжанкова А.С.						6
Утв.	Пацей Н.В.					74417006, 2021	

Таблица 5.1 – Описание юнит-тестов

Модуль	Название теста	Описание
Сервис аутентификации и авторизации	<i>Login_ReturnsTokens</i>	Проверка входа пользователя в систему с возвращением токена доступа и токена обновления
Сервис категорий	<i>Create_ReturnsNewCategory</i>	Проверка создания новой категории с возвращением новой категории
Сервис обмена валют	<i>GetBySourceAndTarget</i>	Проверка возвращения курса обмен валют по двум валютам
Сервис транзакций	<i>Create_ThrowsExceptionAboutBalance</i>	Проверка выбрасывания исключения при создании транзакции с недостаточным балансом
Сервис повторяющихся транзакций	<i>Delete_ReturnsDeletedTransaction</i>	Проверка удаления повторяющейся транзакции с возвращением удаленной транзакции
Сервис типа транзакций	<i>Create_ThrowsException</i>	Проверка выбрасывания исключения при попытке создания нового типа транзакций
Сервис подтверждения кода по почте	<i>Update_ReturnsNewCode</i>	Проверка обновления кода, при котором для пользователя еще не было создано кода до этого
Сервис уведомлений	<i>GetUnreadCount_ReturnsCount</i>	Проверка возвращения количества непрочитанных уведомлений пользователя
Сервис токенов обновления	<i>Update_ReturnsUpdatedToken</i>	Проверка обновления токена с возвращением обновленного токена
Сервис пользователей	<i>UpdatePassword_ReturnsUpdatedUser</i>	Проверка обновления пароля пользователя с возвращением обновленного пользователя
Сервис пользовательских настроек	<i>GetByUnit-Name_ReturnsGeneralSettings</i>	Проверка возвращения настроек пользователя по имени раздела с возвращением основных настроек

Окончание таблицы 5.1

Модуль	Название теста	Описание
Сервис кошельков	<i>CreateTransfer_ThrowsErrorAboutMoney</i>	Проверка выбрасывания исключения о балансе при создании перевода

Таким образом, в результате тестирования, были протестированы все сервисы приложения, включая сервисы *Salesforce* организации.

5.2 Ручное тестирование

Для ручного тестирования были проверены сценарии регистрации и входа в аккаунта, а также создание транзакции и перевод средств с одного кошелька на другой.

5.2.1 Регистрация и вход в аккаунт

В случае формы входа, при ошибке в написании электронного адреса или пароля, выдается сообщение, показанное на рисунке 5.1.

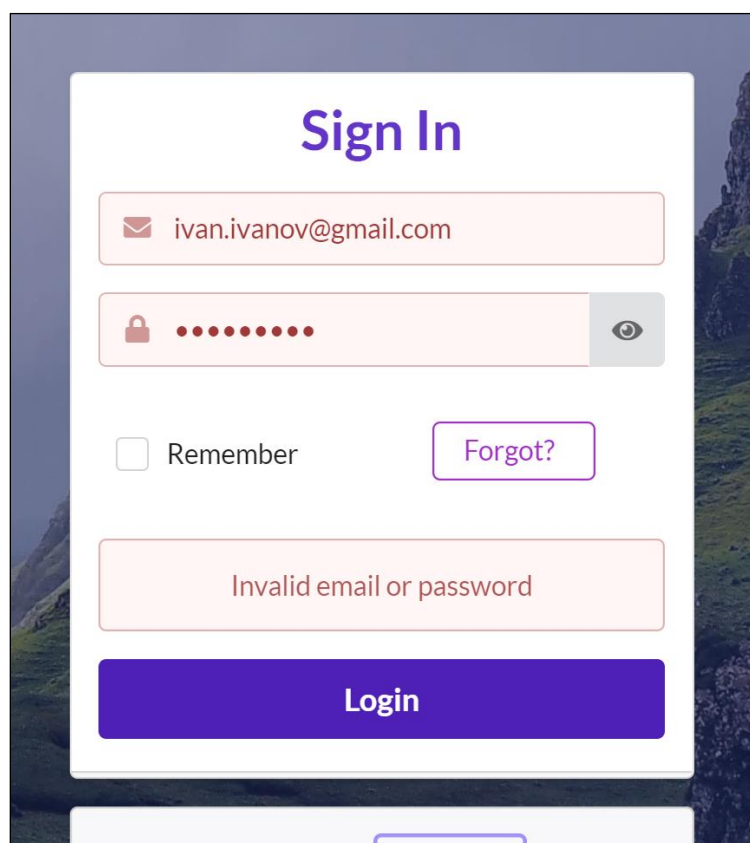
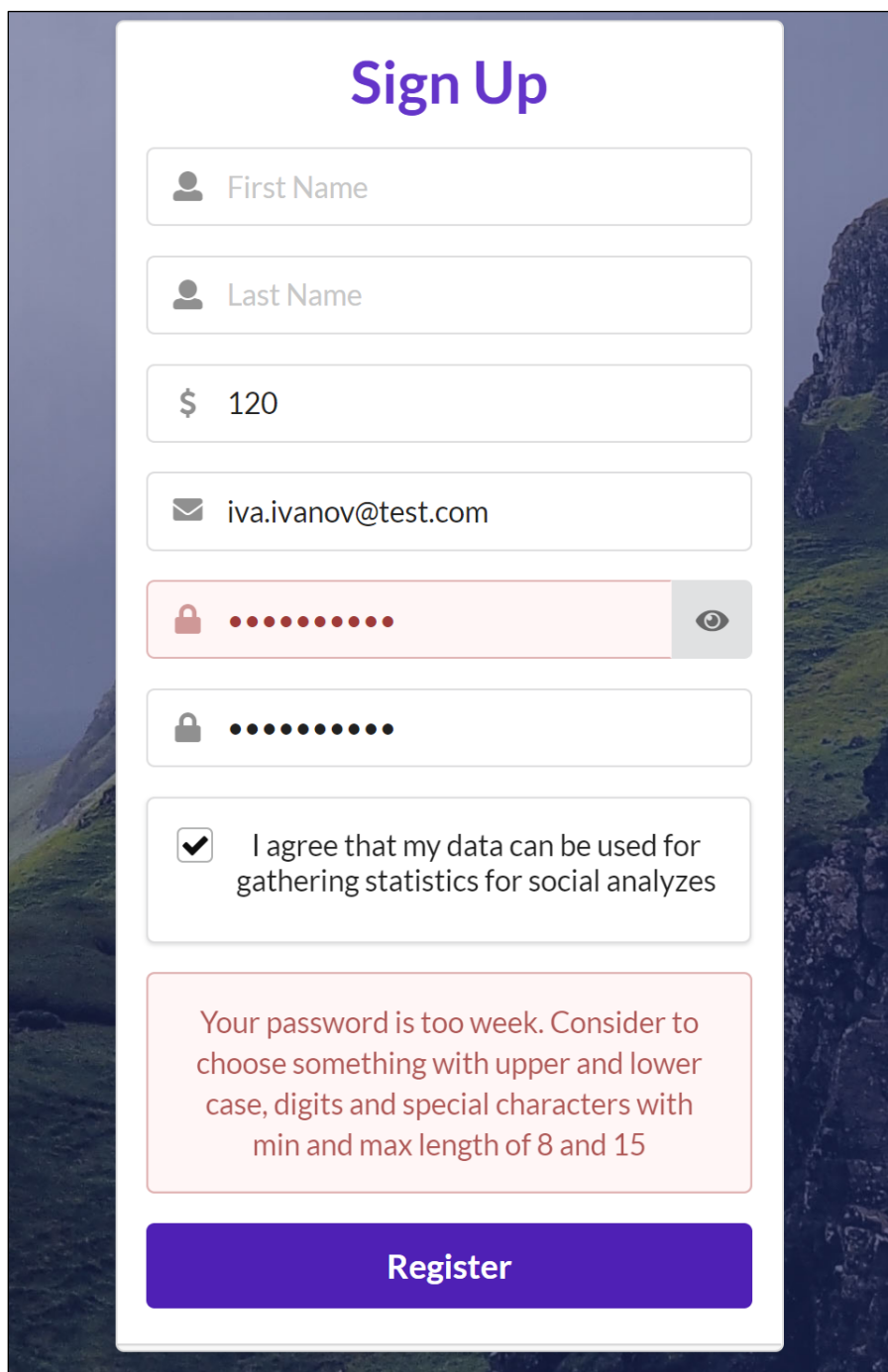


Рисунок 5.1 – Сообщение об ошибке входа в аккаунт


В сообщении пользователю говорится, что введенные им почта или пароль неправильные.


При рассмотрении формы регистрации можно попытаться ввести простой пароль. Результат этого показан на рисунке 5.2.




The image shows a 'Sign Up' form with a purple header. The form contains several input fields: 'First Name', 'Last Name', a currency field with '\$' and '120', and an email field with 'iva.ivanov@test.com'. There are two password fields; the top one is highlighted in red and has a red padlock icon, indicating it is weak. Below the password fields is a checkbox that is checked, with the text 'I agree that my data can be used for gathering statistics for social analyzes'. A red-bordered box contains a message: 'Your password is too week. Consider to choose something with upper and lower case, digits and special characters with min and max length of 8 and 15'. At the bottom is a purple 'Register' button.


Sign Up


 First Name


 Last Name

\$ 120

 iva.ivanov@test.com







☒ I agree that my data can be used for gathering statistics for social analyzes

Your password is too week. Consider to choose something with upper and lower case, digits and special characters with min and max length of 8 and 15

Register

Рисунок 5.2 – Сообщение при попытке ввода простого пароля при регистрации

При попытке регистрации в таком случае пользователь получает сообщение о том, что введенный им пароль слишком слабый и ему нужно использовать для этого большие и маленькие буквы, цифры и специальные символы.

5.2.2 Создание транзакции

Для проверки создания транзакций в данном разделе будет выделен случай, при котором пользователь будет пытаться создать транзакцию без достаточного количества средств на кошельке, к которой она привязана. Для этого сначала выберем кошелек, для которого будем создавать транзакцию. Список кошельков представлен на рисунке 5.3.

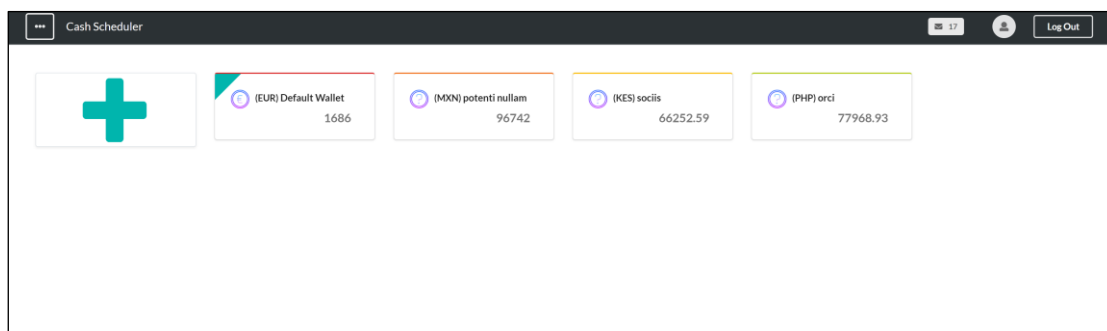


Рисунок 5.3 – Список созданных кошельков

Для теста возьмем кошелек в евро, находящийся первым в списке и помеченный как кошелек по умолчанию. Теперь перейдем на главную страницу и попытаемся создать транзакцию расхода на сегодняшний день, указав сумму транзакции больше суммы, доступной на кошельке, например 1800. Теперь попробуем создать транзакцию. Результат представлен на рисунке 5.4.

Рисунок 5.4 – Заполненная форма новой транзакции

Как мы видим, система не дает нам создать транзакцию с суммой больше, чем есть у нас на кошельке. Таким образом система отработала верно.

Блок-схема алгоритма создания транзакции представлена в приложении Д.

5.2.3 Создание перевода

Для проверки создания перевода средств между двумя кошельками был выбран сценарий при котором пользователь попытается создать перевод с одного кошелька на другой, без достаточного количества средств на исходном кошельке. Для этого были созданы кошельки в разных валютах, для того, чтобы немного усложнить сценарий. Список кошельков представлен на рисунке 5.5.

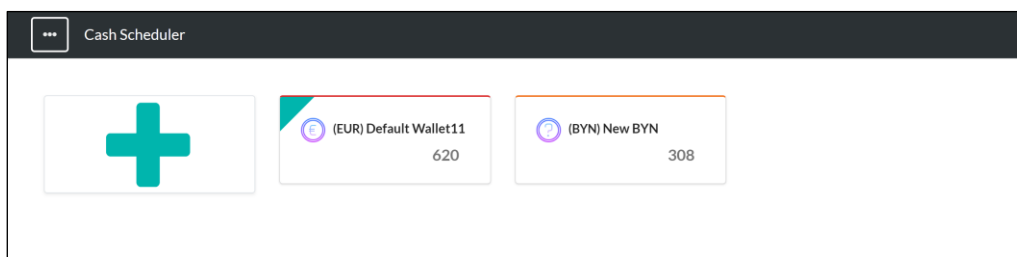


Рисунок 5.5 – Список созданных кошельков

Для тестирования были взяты кошельки в евро и белорусских рублях соответственно. Перетянув один кошелек на другой мы получим форму перевода. Заполнив форму со всей необходимой информацией, введем сумму перевода большую, чем баланс исходного кошелька. Результат показан на рисунке 5.6.

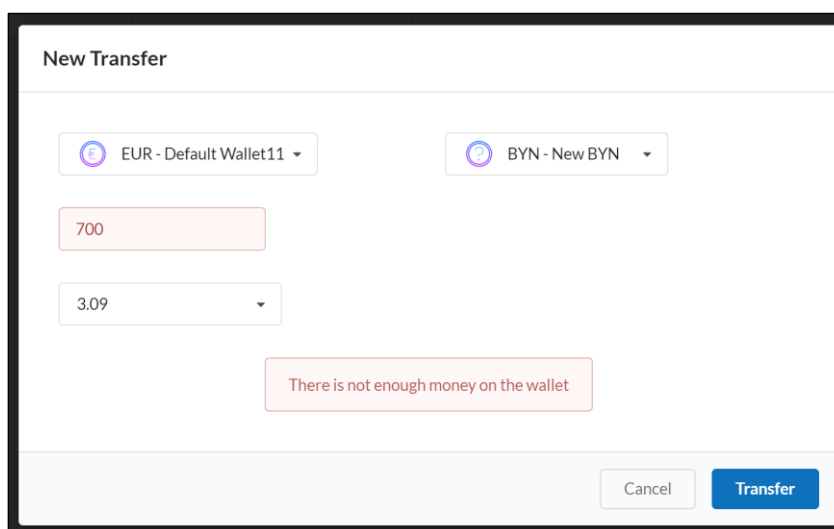


Рисунок 5.6 – Заполненная форма перевода

Как можно увидеть на рисунке, при попытке перевода пользователю была выведена ошибка, сообщающая о том, что баланс на исходном кошельке недостаточный для перевода средств. Таким образом можно сделать вывод, что система отработала верно.

Блок-схема алгоритма создания перевода представлена в приложении Е.

5.3 Вывод по разделу

В данном разделе предоставлены результаты тестирования веб-приложения, разработанного для управления личными финансами. Использовались два вида тестирования, а именно, модульное и ручное. Были написаны тесты для всех сервисов *ASP.NET Core* приложения, а также покрыт весь код *Salesforce* организации. Были осуществлены проверки на ввод некорректных и неправильных данных, а также проверена работоспособность веб-приложения на основные случаи использования.

Полученные результаты при модульном и ручном тестировании показали, что веб-приложение работает правильно и корректно.

6 Экономическое обоснование цены веб-приложения

6.1 Общая характеристика разрабатываемого веб-приложения

При выполнении данного дипломного проекта, была разработана система, представленная веб-приложением. Это приложение состоит из трех основных частей: клиентская, серверная и *Salesforce* организация.

Приложение на начальном этапе будет распространяться для бесплатного пользования.

Данный раздел служит для определения затрат, произведенных на всех стадиях разработки веб-приложения. Также необходимо провести расчет экономии основных видов ресурсов в связи с использованием данного веб-приложения.

6.2 Исходные данные и маркетинговый анализ

Исходные данные для расчета приведены в таблице 6.1.

Таблица 6.1 – Исходные данные для расчета

Наименование показателя	Единица измерения	Условные обозначения	Норматив
Численность разработчиков	чел.		1
Норматив дополнительной заработной платы	%	$H_{дз}$	15
Ставка отчислений в Фонд социальной защиты населения	%	$H_{фсзн}$	34
Ставка отчислений в БРУСП «Белгосстрах»	%	$H_{бгс}$	0,4
Цена одного машино-часа	руб.	$C_{мч}$	0,06
Норматив прочих затрат	%	$H_{пз}$	26,5
Норматив накладных расходов	%	$H_{обп, обх}$	186
Норматив расходов на сопровождение и адаптацию	%	$H_{рса}$	17

В ходе проведения маркетингового анализа, была выявлена стоимость разработки программного продукта для управления финансами. В таблице 6.2 представлена стоимость разработки аналогичных приложений.

					БГТУ 06.00.ПЗ								
Изм	Лист	№ докум.	Подп.	Дата									
Разраб.	Мацуев И.М.				6 Экономическое обоснование цены веб-приложения				Лит.	Лист	Листов		
Пров.	Северинчик Н.А.										1	7	
Консульт.	Евлаш А.И.								74417006, 2021				
Н. контр.	Рыжанкова А.С.												
Утв.	Пацей Н.В.												

Таблица 6.2 – Стоимость разработки аналогичных приложений

Наименование приложения	Описание приложения	Стоимость разработки, руб
<i>MoneyLover</i>	Приложение для планирования и учета расходов и доходов. Дает возможность просмотра статистики транзакций	45 000
<i>Spendee</i>	Приложение для планирования и учета расходов	24 000
<i>Wallet</i>	Приложение для планирования и учета расходов и доходов. Дает возможность привязки банковских карт	36 000

Средняя цена разработки аналогичного продукта составляет 30 000 – 35 000 рублей. Таким образом, примерная общая стоимость разработки данного веб-приложения, выбранного в качестве базы сравнения составит 34 000 рублей.

6.3 Методика обоснования цены

В современных рыночных экономических условиях веб-приложения выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид, реализуемые покупателям по рыночным отпускным ценам

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования приложений. Выбор эффективных проектов связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации приложения, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

- снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
- сокращения расходов на оплату машинного времени и других ресурсов на отладку программ;
- снижения расходов на материалы;
- ускорение ввода в эксплуатацию новых систем;
- улучшения показателей основной деятельности в результате использования веб-приложения.

Стоимостная оценка приложения у разработчиков предполагает определение затрат, что включает следующие статьи:

- заработная плата исполнителей – основная и дополнительная;
- отчисления в фонд социальной защиты населения;

- отчисления по обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
- расходы на материалы и комплектующие;
- расходы на спецоборудование;
- расходы на оплату машинного времени;
- прочие прямые затраты.

На основании затрат рассчитывается себестоимость и отпускная цена приложения.

6.3.1 Объем веб-приложения

Для оценки объема веб-приложения, все его функции классифицируются с использованием специального каталога функций, который определяет их объем. Общий объем приложения V_o , вычисляется как сумма объемов V_i каждой из n его функций (формула 6.1).

$$V_o = \sum_{i=1}^n V_i \quad (6.1)$$

В таблице 6.3 представлены функции, присутствующие в рассматриваемом веб-приложении и соответствующий им объем в условных машино-командах.

Таблица 6.3 – Содержание и объем функций в приложении

Номер функции	Содержание функции	Объем, условных машино-команд
101	Организация ввода информации	720
102	Контроль, предварительная обработка и ввод информации	450
111	Управление вводом/выводом	1800
202	Взаимодействие между компонентами системы	2400
401	Взаимодействие с базой данных	150
402	Вспомогательные методы	330
506	Обработка ошибочных и сбойных ситуаций	960
707	Графический вывод результатов	1700
	Итого:	8510

Опираясь на данные таблицы 6.3, можно определить объем веб-приложения, разработанного в ходе дипломного проектирования:

$$V_o = 720 + 450 + 1800 + 2400 + 150 + 330 + 960 + 1700 = 8510 \text{ (условных машино-команд)}.$$

Уточненный объем веб-приложения V_o' равен произведению объема приложения V_o на коэффициент изменения скорости обработки информации $K_{ск}$ (формула 6.2).

$$V_o' = V_o \cdot K_{ск} . \quad (6.2)$$

Исходя из вычисленного объема веб-приложения, можно определить его уточненный объем:

$$V_o' = 8510 \cdot 0,6 = 5106 \text{ (условных машино-команд).}$$

6.3.2 Основная заработная плата

Для определения величины основной заработной платы, было проведено исследование величин заработных плат для специалистов в сфере веб-программирования на *ASP.NET Core*. Источником данных служили открытые веб-порталы, различные форумы, официальная отчетность, а также общий средний уровень заработка в сфере информационных технологий. Итогом изучения и анализа полученных данных, стала информация о том, что средняя месячная заработная плата для позиций junior/middle составляет 1800 рублей.

Проект разрабатывался одним человеком на протяжении трех месяцев. Таким образом, основная заработная плата будет рассчитываться по формуле (6.3):

$$C_{оз} = T_{раз} \cdot K_{раз} \cdot C_{зп}, \quad (6.3)$$

где $C_{оз}$ – основная заработная плата, руб.;

$T_{раз}$ – время раработки, месяцев;

$K_{раз}$ – количество разработчиков, человек;

$C_{зп}$ – средняя месячная заработная плата.

$$C_{оз} = 3 \cdot 1 \cdot 1800 = 5400 \text{ руб.}$$

6.3.3 Дополнительная заработная плата

Законодательство о труде предусматривает наличие выплат, которые определяются по нормативу в процентах к основной заработной плате по формуле (6.4):

$$C_{дз} = \frac{C_{оз} \cdot H_{дз}}{100}, \quad (6.4)$$

где $C_{оз}$ – основная заработная плата, руб.;

$H_{дз}$ – норматив дополнительной заработной платы, %.

$$C_{дз} = 5400 \cdot 15 / 100 = 810 \text{ руб.}$$

6.3.4 Отчисления в Фонд социальной защиты населения

Отчисления в Фонд социальной защиты населения (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле (6.5):

$$C_{\text{фсзн}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot N_{\text{фсзн}}}{100}, \quad (6.5)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$C_{\text{дз}}$ – дополнительная заработная плата, руб.;

$N_{\text{фсзн}}$ – норматив отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 6.6:

$$C_{\text{бгс}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot N_{\text{бгс}}}{100}, \quad (6.6)$$

$$C_{\text{фсзн}} = (5400 + 810) \cdot 34 / 100 = 2111,4 \text{ руб.}$$

$$C_{\text{бгс}} = (5400 + 810) \cdot 0,4 / 100 = 24,84 \text{ руб.}$$

6.3.5 Расходы на материалы

Сумма расходов на материалы $C_{\text{м}}$ определяется как произведение нормы расхода материалов в расчете на сто строк исходного кода $N_{\text{м}}$ на уточненный объем веб-приложения V_o' (формула 6.7).

$$C_{\text{м}} = N_{\text{м}} \cdot \frac{V_o'}{100}. \quad (6.7)$$

Учитывая, что норма расхода материалов в расчете на сто строк исходного кода равен 0,46 руб., можно определить сумму расходов на материалы:

$$C_{\text{м}} = 0,46 \cdot 5106 / 100 = 23,5 \text{ руб.}$$

6.3.6 Расходы на оплату машинного времени

Сумма расходов на оплату машинного времени $C_{\text{мв}}$ определяется как произведение стоимости одного машино-часа $C_{\text{мч}}$ на уточненный объем веб-приложения V_o' и на норматив расхода машинного времени на отладку ста строк исходного кода $N_{\text{мв}}$ (формула 6.8).

$$C_{\text{мв}} = C_{\text{мч}} \cdot \frac{V_o'}{100} \cdot N_{\text{мв}}. \quad (6.8)$$

Учитывая, что норматив машинного времени на отладку ста строк исходного кода равен 12, можно определить сумму расходов на оплату машинного времени:

$$C_{\text{мв}} = 0,06 \cdot 5106 \cdot 12 / 100 = 36,76 \text{ руб.}$$

6.3.7 Прочие прямые затраты

Сумма прочих затрат $C_{\text{пз}}$ определяется как произведение основной заработной платы исполнителей на конкретное веб-приложение $C_{\text{оз}}$ на норматив прочих затрат в целом по организации $H_{\text{пз}}$ (формула 6.9).

$$C_{\text{пз}} = \frac{C_{\text{оз}} \cdot H_{\text{пз}}}{100}, \quad (6.9)$$

$$C_{\text{пз}} = 5400 \cdot 26,5 / 100 = 1431 \text{ (руб.)}.$$

6.3.8 Накладные расходы

Сумма накладных расходов $C_{\text{обп,обх}}$ – произведение основной заработной платы исполнителей на конкретное веб-приложение $C_{\text{оз}}$ на норматив накладных расходов в целом по организации $H_{\text{обп,обх}}$ (формула 6.10).

$$C_{\text{обп,обх}} = \frac{C_{\text{оз}} \cdot H_{\text{обп,обх}}}{100}, \quad (6.10)$$

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

$$C_{\text{обп,обх}} = 5400 \cdot 186 / 100 = 10\,044 \text{ руб.}$$

6.3.9 Сумма расходов на разработку веб-приложения

Сумма расходов на разработку веб-приложения $C_{\text{р}}$ определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное приложение, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов (формула 6.11).

$$C_{\text{р}} = C_{\text{оз}} + C_{\text{дз}} + C_{\text{фсзн}} + C_{\text{бгс}} + C_{\text{м}} + C_{\text{мв}} + C_{\text{пз}} + C_{\text{обп,обх}}, \quad (6.11)$$

$$C_{\text{р}} = 5400 + 810 + 2111,4 + 24,84 + 23,5 + 36,76 + 1431 + 10\,044 = 19\,881,5 \text{ руб.}$$

6.3.10 Расходы на сопровождение и адаптацию

Сумма расходов на сопровождение и адаптацию веб-приложения $C_{\text{рса}}$ определяется как произведение суммы расходов на разработки на норматив расходов на сопровождение и адаптацию $H_{\text{рса}}$ (формула 6.12).

$$C_{\text{рса}} = \frac{C_{\text{р}} \cdot H_{\text{рса}}}{100}, \quad (6.12)$$

$$C_{pca} = 19\,881,5 \cdot 17 / 100 = 3379,86 \text{ (руб.)}.$$

6.3.11 Полная себестоимость

Полная себестоимость C_{π} определяется как сумма двух элементов: суммы расходов на разработку C_p и суммы расходов на сопровождение и адаптацию веб-приложения C_{pca} (формула 6.13).

$$C_{\pi} = C_p + C_{pca}, \quad (6.13)$$

$$C_{\pi} = 19\,881,5 + 3379,86 = 23\,261,36 \text{ (руб.)}.$$

Полная себестоимость веб-приложения была вычислена на основе данных, рассчитанных ранее в данном разделе.

6.4 Вывод по разделу

В таблице 6.4 представлены результаты расчетов для основных показателей данного раздела в краткой форме.

Таблица 6.4 – Результаты расчетов

Наименование показателя	Значение
Время разработки, мес.	3
Количество программистов, чел.	1
Зарплата с отчислениями, руб.	8346,24
Расходы на материалы, оплату машинного времени, прочие, руб	1491,26
Накладные расходы, руб	10 044
Себестоимость разработки веб-приложения, руб.	19 881,5
Расходы на сопровождение и адаптацию, руб.	3379,86
Полная себестоимость, руб.	23 261,36
Цена аналога, руб.	34 000

Необходимость разработки приложения обусловлена ростом уровня жизни людей, что создает проблемы по части мониторинга и фильтрации источников доходов и расходов граждан. Чтобы решить данную проблему было веб-приложение, которое поможет пользователям фиксировать их траты и прибыль.

Разработка веб-приложения, осуществляемая одним программистом в течении трех месяцев, при заданных условиях обойдется компании в 23 261,36 руб. По сравнению с изученными аналогами примерная экономия на разработке составит 10 739,14 руб.

Заключение

В ходе выполнения дипломного проекта было разработано веб-приложение для управления личными финансами, которое имеет следующие возможности:

- регистрацию пользователя;
- восстановление пароля пользователя;
- авторизация пользователя;
- изменение информации о пользователе;
- создание и управление кошельками;
- создание и управление категориями;
- создание и управление одиночными и повторяющимися транзакциями;
- отправка отзывов и обратной связи;
- визуализация статистики транзакций пользователя за период времени;
- настройка параметров работы приложения для пользователя;
- механизм уведомлений внутри приложения;
- сбор и визуализация информации об использовании приложения.

Также были проанализированы и выбраны основные технологии и средства для разработки дипломного проекта. Была спроектирована база данных и выбрана наиболее подходящая структура проекта.

Веб-приложение было реализовано на языке *C#* на платформе *ASP.NET Core* с использованием *React JS* для написания клиентской части. Дополнительно была создана, настроена и интегрирована *Salesforce* организация. В качестве программного веб-интерфейса использовался язык запросов *GraphQL* с применением библиотеки *Hot Chocolate* для *ASP.NET Core*. Для работы с *GraphQL* на клиентской части использовалась библиотека *Apollo Client*, как альтернатива *Redux*.

Результатом является законченное веб-приложение, хотя возможна доработка.

Пользование данным приложением не составит труда для людей любого возраста, так как оно имеет удобный и понятный интерфейс.

Данное приложение соответствует поставленной задаче и отвечает всем требованиям, необходимым для его удобного использования.

					БГТУ 00.00.ПЗ									
Изм	Лист	№ докум.	Подп.	Дата										
Разраб.	Мацуев И.М.				Заключение					Лит.	Лист	Листов		
Пров.	Северинчик Н.А.											1	1	
Консульт.	Северинчик Н.А.									74417006, 2021				
Н. контр.	Рыжанкова А.С.													
Утв.	Пацей Н.В.													

Список использованных источников

- 1 Money Lover. Simplest way to manage personal finances [Электронный ресурс]. – Режим доступа: <https://moneylover.me/>. – Дата доступа: 18.03.2021.
- 2 Spendee. The only app that gets your money into shape [Электронный ресурс]. – Режим доступа: <https://www.spendee.com/>. – Дата доступа: 19.03.2021.
- 3 Wallet. Your Finances in One Place [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/dotnet/csharp/>. – Дата доступа: 20.03.2021.
- 4 C# documentation. Get Started [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/dotnet/csharp/>. – Дата доступа: 27.03.2021.
- 5 ASP.NET documentation. Get Started [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-5.0/>. – Дата доступа: 28.03.2021.
- 6 Работа с Entity Framework 6 [Электронный ресурс]. – Режим доступа: <https://professorweb.ru/my/entity-framework/6/level1/>. – Дата доступа: 01.04.2021.
- 7 Тестирование. Введение в юнит-тесты [Электронный ресурс]. – Режим доступа: <https://metanit.com/sharp/aspnet5/22.1.php/>. – Дата доступа: 15.04.2021.
- 8 Hot Chocolate. Introduction [Электронный ресурс]. – Режим доступа: <https://chillicream.com/docs/hotchocolate/v10/>. – Дата доступа: 29. 03. 2021.
- 9 React [Электронный ресурс]. – Режим доступа: <https://ru.reactjs.org/>. – Дата доступа: 02.04.2021.
- 10 Semantic UI [Электронный ресурс]. – Режим доступа: <https://react.semantic-ui.com/usage/>. – Дата доступа: 03.04.2021.
- 11 Introduction to Apollo Client [Электронный ресурс]. – Режим доступа: <https://www.apollographql.com/docs/react/>. – Дата доступа: 10. 04. 2021.
- 12 The React + Apollo Tutorial [Электронный ресурс]. – Режим доступа: <https://www.freecodecamp.org/news/react-apollo-client-2020-tutorial/>. – Дата доступа: 13.04.2021.
- 13 Omni-Channel Readiness and Digital Engagement [Электронный ресурс]. – Режим доступа: <https://trailhead.salesforce.com/en/content/learn/modules/omni-channel-readiness-and-digital-engagement/>. – Дата доступа: 16. 04. 2021.
- 14 Что находится между идеей и кодом? Обзор 14 диаграмм UML [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/508710>. – Дата доступа: 20.04.2021.
- 15 Гайд по ручному тестированию приложений [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/company/skillbox/blog/418889>. – Дата доступа: 07.05.2021.
- 16 Калькулятор стоимости разработки приложения [Электронный ресурс]. – Режим доступа: <https://appcraft.pro/blog/kalkulyator-stoimosti-mobilnogo-prilozheniya/>. – Дата доступа: 16.05.2021.

					БГТУ 00.00.ПЗ								
Изм	Лист	№ докум.	Подп.	Дата									
Разраб.	Мацуев И.М.				Список использованных источников				Лит.	Лист	Листов		
Пров.	Северинчик Н.А.										1	1	
Консульт.	Северинчик Н.А.								74417006, 2021				
Н. контр.	Рыжанкова А.С.												
Утв.	Пацей Н.В.												

ПРИЛОЖЕНИЕ А
Диаграмма использования

ПРИЛОЖЕНИЕ Б
Логическая схема базы данных

ПРИЛОЖЕНИЕ В
Диаграмма классов

ПРИЛОЖЕНИЕ Г
Скриншоты работы программы

ПРИЛОЖЕНИЕ Д
Блок-схема алгоритма создания транзакции

ПРИЛОЖЕНИЕ Е
Блок-схема алгоритма создания перевода